FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DERİN ÖĞRENME ALGORİTMASI KULLANAN BİR MOBİL ABARTILMIŞ GERÇEKLİK OYUNU

YÜKSEK LİSANS TEZİ

UMUT BOZ

tarafından

YÜKSEK LİSANS

derecesi şartını sağlamak için hazırlanmıştır.

Nisan 2019

Program: Bilgisayar Mühendisliği

DERİN ÖĞRENME ALGORİTMASI KULLANAN BİR MOBİL ABARTILMIŞ GERÇEKLİK OYUNU

YÜKSEK LİSANS TEZİ

UMUT BOZ

tarafından

İSTANBUL OKAN ÜNİVERSİTESİ

Bilgisayar Mühendisliği Anabilim Dalına

Yüksek Lisans

derecesi şartını sağlamak için sunulmuştur.

Onaylayan

Danışman
Prof. Dr. Bekir Tevfik AKGÜN

Jüri Üyesi
Doç. Dr. Pınar YILDIRIM

Jüri Üyesi
Doç. Dr. Erchan APTOULA
(Gebze Teknik Üniversitesi)

Nisan 2019

Program: Bilgisayar Mühendisliği

ÖZET

Bu tezde; Derin Öğrenme, Bilgisayarlı Görü, Nesne Sınıflandırma, Nesne Belirleme ve Görüntü İşleme teknikleri kullanarak bir Artırılmış Gerçeklik (AG) uygulaması gerçekleştirilmiştir.

Bu yeni nesil teknolojilerle desteklenen ve gerçeklenen mobil AG oyunu ile sadece tek kamera kullanarak; sanal ortam üzerinde, bir fiziksel nesne ile bir sanal nesnenin fiziksel dünya kurallarının yer aldığı etkileşiminin sağlanması amaçlanmıştır. Bu çalışma örnek alınarak, gerçek dünyaya ait nesneler üzerinde ilgili örnekler çoğaltılarak AG dünyasında farklı uygulamalar ile çözümler üretilebilinir. Üretim sektörü, eğitim, oyun, çocuk gelişimi ve benzeri alanlara yönelik kamera üzerinden öğretilmiş nesneleri kullanan uygulamalar gerçekleştirilebilinir.

Genel mobil cihaz sensörlerine karşın; 3 boyutlu nesneleri algılayan sensörler, donanımlar ve özel kameralar kullanmak şüphesiz daha gerçeğe uygun bir çalışma ortamı sunar. Ancak son kullanıcılar açısından bakılırsa; bu tezde önerdiğimiz yöntemleri kullanan mobil AG çözümleri fazladan bir maliyet oluşturmayacağından daha yaygın kullanım alanı bulacaktır.

Anahtar Kelimeler: Artırılmış Gerçeklik, ARKit, Bilgisayarlı Görü, Örüntü Tanıma, Makine Öğrenimi, Derin Öğrenme, Mobil Programlama, Nesne Tespiti

ABSTRACT

In this thesis, an Augmented Reality (AR) application was carried out by using Deep Learning, Computer Vision, Object Detection, Object Classification and Image Processing techniques.

Using only one camera with the mobile AR game supported by the next generation technologies mentioned above; it is aimed to provide the interaction of a physical object and a virtual object on the virtual environment with the physical world rules. Based on this work, solutions can be produced with different applications in the world of AR by duplicating related examples on real world objects. Applications using objects taught through the camera for areas such as production, education, play, child development and the like can be realized.

Despite general mobile device sensors; The use of sensors, hardware and special cameras that detect 3D objects offers a more realistic working environment. But from the perspective in terms of end users; AR solutions that use the methods we propose in this thesis will be used more widely while they will not constitute an extra cost.

Keywords: Augmented Reality, ARKit, Computer Vision, Pattern Recognition, Machine Learning, Deep Learning, Mobile Development, Object Detection

DERİN ÖĞRENME ALGORİTMASI KULLANAN

BİR MOBİL ABARTILMIŞ GERÇEKLİK OYUNU

TEŞEKKÜR

Çalışma süresince bana desteğini esirgemeyen, bilgi birikimleri ile çalışmama ışık tutan, deneyimlerini benimle paylaşan tez danışman hocam Sayın Prof. Dr. Bekir Tevfik AKGÜN'e teşekkür ederim.

Bu çalışma boyunca yardımlarını ve manevi desteklerini esirgemeyen aileme, yakınlarıma ve arkadaşlarıma şükranlarımı sunarım.

Umut BOZ

Nisan 2019

İÇİNDEKİLER

ÖZET II

ABSTRACT III

TABLO LİSTESİ VIII

ŞEKİL LİSTESİ ..... İX

SEMBOLLER ..... Xİİ

I. GİRİŞ 1

II. KULLANILAN TEKNOLOJİLER 8

2.1. DERİN ÖĞRENME 8

2.1.1. Evrişimli Sinir Ağları 9

2.1.1.1. Evrişimli Sinir Ağları Mimarileri 14

2.1.2. Derin Öğrenmede Yazılım ve Donanım Gereksinimleri 15

2.1.3. Mobil Uygulamalarda Derin Öğrenme Teknolojileri 19

2.2. ARTIRILMIŞ GERÇEKLİK 19

2.2.1 Artırılmış Gerçeklik Kütüphaneleri 21

2.3. KULLANILAN PROGRAMLAMA DİLLERİ 22

2.3.1. Python 22

2.3.2. C++ 22

2.3.3. Objective C 23

2.3.4. Swift 23

2.4. KULLANILAN KÜTÜPHANE ÇÖZÜMLERİ 24

2.4.1. TuriCreate 24

2.4.5. Vision 25

2.4.2. CoreML 25

2.4.3. ARKit 26

2.4.4. OpenCV 26

III. SİSTEM TASARIMI 27

3.1. HAZIRLIK EVRESİ TASARIMI 27
3.2. KULLANICI İLE ETKİLEŞİM EVRESİ TASARIMI 27
3.2.1. Bardak Tanımlama Tasarımı 28
3.2.2. Bardak Ağızı Belirleme Tasarımı 30
3.2.3. Bardak Boyutu ve Konum Belirleme Tasarımı 31
3.2.4. Oyun Gerçekleme Tasarımı 32
3.3. OYUNUN OYNANMASI 33

IV. SİSTEM GERÇEKLEMESİ 35

4.1. HAZIRLIK EVRESİ GERÇEKLEMESİ 35
4.1.1. Veri Toplama Gerçeklemesi 35
4.1.2. Görüntü Ön İşleme Gerçeklemesi 37
4.1.3. Eğitim Verisi Etiketleme ve Çıkarma Gerçeklemesi 40
4.1.4. Eğitimin Gerçeklenmesi 42
4.1.5. Test Gerçeklemesi 46
4.1.6. Mobil Adaptasyonu Gerçeklemesi 48
4.2. KULLANICI İLE ETKİLEŞİM EVRESİ GERÇEKLEMESİ 49
4.2.1. Bardak Tanımlama Gerçeklemesi 50
4.2.2. Bardak Ağzı Belirleme Gerçeklemesi 56
4.2.3. Bardak Boyutu ve Konumu Gerçeklemesi 60
4.2.4. Oyun Oynama Evresinin Gerçeklemesi 62

V. SONUÇLAR 66

VI. KAYNAKLAR 68

TABLO LİSTESİ

Tablo II.1 ESA Mimarileri 14
Tablo II.2 Derin Öğrenme Kütüphaneleri ve Özellikleri 16
Tablo II.3 Artırılmış Gerçeklik Kütüphaneleri 21
Tablo II.4 Kullanılan C++ Paketleri 23
Tablo II.5 Kullanılan Swift Paketleri 24
Tablo IV.1 Elips Parametreleri 57
Tablo IV.2 İvme Ölçer Konumları 58

ŞEKİL LİSTESİ

Şekil I.1 Büyülü Kılıç cihazı 1
Şekil I.2 Sanal Fikstür cihazı 2
Şekil I.3 Google Glass 3
Şekil I.4 Ikea Place uygulaması 4
Şekil I.5 Karma Gerçeklik donanımları 5
Şekil II.1 Bilgisayar Görüşü 11
Şekil II.2 Filtreleme ile özellik haritasının çıkartılması 12
Şekil II.3 ESA'ların farklı katmanlarda gerçekleşen filtreleme işlemleri [9] 12
Şekil II.4 Evrişim İşlemi ve filtreleme işlemleri 13
Şekil II.5 Evrişimli Sinir Ağı modeli ilerleyişi 13
Şekil II.6 Derin Öğrenmede tercih edilen programlama dili istatistikleri 16
Şekil II.7 SLAM 2 boyutlu özellik noktaları 21
Şekil II.8 CoreML model seviyesi 26
Şekil III.1 Hazırlık evresi tasarımı 27
Şekil III.2 Kullanıcı ile etkileşim tasarımı 28
Şekil III.3 Bardak tanımlama tasarımı 29
Şekil III.4 Bardak ağızı belirleme evresi tasarımı 30
Şekil III.5 Bardak boyutu ve konum belirleme evresi tasarımı 31
Şekil III.6 Oyun gerçekleme tasarımı 32
Şekil III.7 Oyun özeti bardak tespiti 33
Şekil III.8 Oyun özeti bardak konumu ve boyutu 34
Şekil III.9 Oyun özeti topun gösterimi 34
Şekil IV.1 ImageNet görsel temini 35
Şekil IV.2 Görsel bağlantıları 36
Şekil IV.3 Görsel bağlantılarını indiren Python kod bloğu 36
Şekil IV.4 Görselleri yeniden adlandıran Python kod bloğu 37
Şekil IV.5 Görselleri yeniden boyutlandıran Python kod bloğu 38
Şekil IV.6 224x224 olarak yeniden boyutlandıran Python kod bloğu 39
Şekil IV.7 Yeniden adlandırılmış ve ölçeklendirilmiş görseller 39
Şekil IV.8 VIA aracı üzerinde yapılan bir bölge tespiti 40

Şekil IV.9 VIA aracı üzerinde birden fazla nesneye ait bölge belirleme 41
Şekil IV.10 VIA aracı üzerinden json çıktısı 41
Şekil IV.11 Nesne Tespiti için Python paketleri 43
Şekil IV.12 .json içerisindeki X1 değeri kök düğümü 43
Şekil IV.13 Nesne Tespitinde görsel ile bölge tespiti verisini eşleştirme 44
Şekil IV.14 Nesne Tespitinde eğitim seti oluşturma 44
Şekil IV.15 Nesne Tespitinde TuriCreate eğitim modeli oluşturma 45
Şekil IV.16 Nesne Tespitinde TuriCreate test işlemi 46
Şekil IV.17 Nesne Tespitinde test çıktısı 47
Şekil IV.18 Nesne Tespitinde TuriCreate explorer() test çıktısı 47
Şekil IV.19 TuriCreate ile birden fazla nesne tespiti test çıktısı 48
Şekil IV.20 TuriCreate Nesne Tespiti modelin CoreML dönüşümü 49
Şekil IV.21 XCode üzerinde dahil edilen eğitim modeli 49
Şekil IV.22 Swift ile kamera görüntüsü alma 50
Şekil IV.23 Swift ile görüntü çözünürlük dönüşümü 51
Şekil IV.24 Swift ile CIImage nesnesi üzerinde oryantasyon ek paketi 51
Şekil IV.25 Swift ile cihaz oryantasyonu alma ek paketi 51
Şekil IV.26 Swift ile CIImage nesnesi üzerinde UIImage ek paketi 52
Şekil IV.27 Eğitmiş olduğumuz modelin kullanım parametreleri 52
Şekil IV.28 Eğitmiş olduğumuz modelin değişken ataması 53
Şekil IV.29 Swift Nesne Tespiti isteği 53
Şekil IV.30 Swift Nesne tespiti çok boyutlu dizisi yanıtı ayıklama 54
Şekil IV.31 TuriCreate nesne tespiti bölge konumlandırması [43]. 54
Şekil IV.32 Swift için bölge konumlandırması çerçevesi dönüşümü 55
Şekil IV.33 Swift tahminleme başarısı filtreleme 55
Şekil IV.34 Swift başarılı tahminleme çerçeve çizimi 56
Şekil IV.35 Daire ve elips eksen dönüşümü 56
Şekil IV.36 Elipsin parametreleri gösterimi 57
Şekil IV.37 Daire içindeki elipse çember denklemi 58
Şekil IV.38 İvme Ölçer sensörü 59
Şekil IV.39 Bardak ağzı çizimi ekran görüntüsü 59
Şekil IV.40 SCNTube nesnesi 60

Şekil IV.41 OpenCV ile bardak merkezinin bulunması 61
Şekil IV.42 Tüp yerleştirilmesi gösterimi..... 62
Şekil IV.43 Top atış şiddeti hesaplaması..... 63
Şekil IV.44 Kullanıcının bardak ile etkileşimi..... 65

SEMBOLLER

$	Dolar
α	Alpha
β	Beta
π	Pi Sayısı
ϕ	Koordinat eksenlerine göre yarım eksen açısı
°F	İvme ölçer açısı
Xe, Ye	Elips noktaları
Xm, Ym	Elips merkezi
r	yarı çap

KISALTMALAR

AG	: Artırılmış Gerçeklik	Augmented Reality
DÖ	: Derin Öğrenme	Deep Learning
MÖ	: Makine Öğrenimi	Machine Learning
ESA	: Evrişimli Sinir Ağları	Convolutional Neural Network
TÖ	: Transfer Öğrenimi	Transfer Learning
KG	: Karma Gerçeklik	Mixed Reality
SG	: Sanal Gerçeklik	Virtual Reality

I. GİRİŞ

Artırılmış Gerçeklik (AG) (Augmented Reality) teknolojisi hızlı gelişen bilişim teknolojileri alanında son zamanlarda en çok ilgi çeken çalışmalardan birisi olmuştur. Sürekli olarak gelişim içinde bulunduğumuz zaman diliminde ise günlük yaşantımızda daha fazla yer almaktadır.

Artırılmış gerçekliği tam olarak yansıtmasada 2. Dünya Savaşı sırasında ve sonrasında askeri projelerde geliştirilen bu alandaki ilk sistemlerin zaman içinde gelişim süreci devam etmiştir.

1960'larda AG ile ilgili ilk örneklere, Ivan Sutherland ve Bob Sproull'un "The Sword of Damocles" adını verdikleri ilkel bilgisayar grafiklerini görüntüleyen ve insan başına takılabilen bir cihaz gösterilebilir. Bu cihaz Şekil I.1'de gösterilmiştir.

Şekil I.1 Büyülü Kılıç cihazı

1970'lerde Myron Krueger, yapay bir gerçeklik laboratuvarı olan Videoplace'i üretti. Dijital şeylerle etkileşimi insan hareketleriyle öngörüyordu. Bu kavram

daha sonra belirli projektörler, video kameralar ve ekranlar için kullanıldı. Seksenlerde ise; Steve Mann, gözlük gibi tasarlanmış EyeTap adında taşınabilir bir bilgisayar geliştirdi. Sahneyi üst üste binen efektlerle kaydetti ve bu görüntüleri kullanıcının baş hareketleriyle oynatabilmesini sağladı. 1987 yılında, Douglas George ve Robert Morris, gerçek gökyüzü üzerinde astronomik veriler gösteren HUD (Heads Up Display) prototipini geliştirdi. 1990'lar AG teknolojilerinin daha hızla geliştiği yıllar olmuştu. AG, bu yıllarda ilk olarak Thomas Caudell ve David Mizell tarafından Boeing şirketi araştırmalarında uygulandı. 1992'de ABD Hava Kuvvetleri'nden Louis Rosenberg "Sanal Fikstür" (Virtual Fixtures) adlı AG sistemini üretti. Sanal Fikstür adlı cihaz Şekil I.2'de gösterilmiştir.

Şekil I.2 Sanal Fikstür cihazı

2000 yılında bir Japon bilim adamı Hirokazu Kato, açık kaynaklı bir SDK olan ARToolKit'i geliştirdi ve yayınladı [1]. 2004 yılında Trimble Navigation, dış mekan için kullanılabilen kask şeklinde bir AG sistemini sundu. 2008 yılında Wikitude firması, Android mobil cihazlar için AG destekli Seyahat Rehberi (AR Travel Guide) uygulamasını yayınladı [2].

Hintli bilgisayar uzmanı Pranav Mistry; bir konuşmasında açık kaynak olarak paylaşacağını belirtmiş olduğu Altıncı His teknolojisiyle, beş duyumuza altıncıyı ekleyerek fiziksel dünyamıza daha az bağımlı olacağımızı ve başka makinelerin karşısında başka makineler olmamamız gerektiğini ifade etmişti [3].

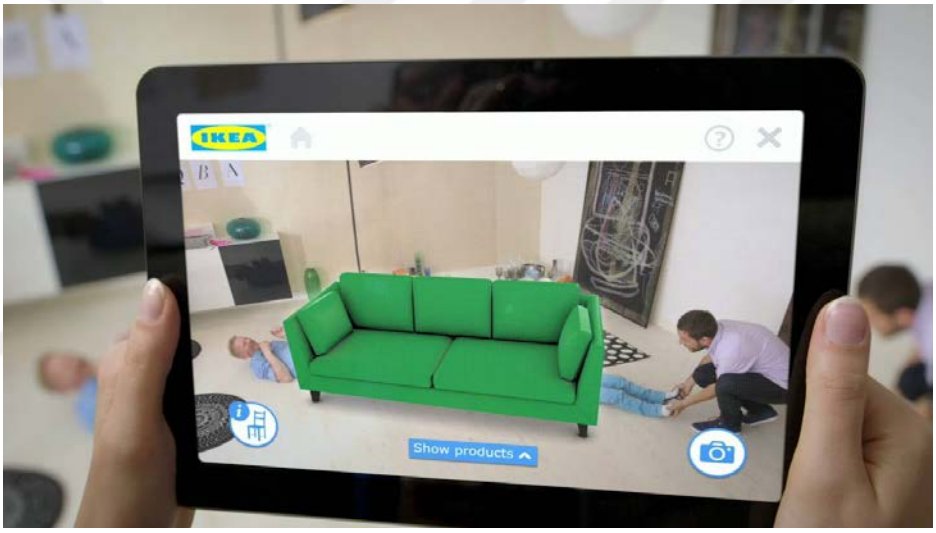
2013 yılında Google, Bluetooth ve internet bağlantısı sağlayabilen Google Glass'ın beta testlerini yaptığını duyurdu. Şekil I.3'te Google Glass cihazı gösterilmektedir.

Şekil I.3 Google Glass

Microsoft, 2015 yılında Windows Holografik ve HoloLens olmak üzere iki yeni teknoloji sundu. Bu çalışma HD hologramları görüntülemek için çok sayıda AG gözlükten oluşmaktaydı.

2016 yılında ise; Niantic firması oyun endüstrisinde çok ses getiren Pokemon Go oyununu duyurdu. Firma, mobil cihazlar için geliştirilen bu oyun ile ilk haftada 2.000.000$ kazandı.

Yine aynı dönemlerde IKEA, mağaza dekoru ve ışıkları altında çok güzel duran kanepelerin ve eşyaların, müşterilerin kendi ev ortamlarında aynı şartları sağlamaması sonucunda hayal kırıklığı yaşamamaları için IKEA Place adını verdiği bir mobil AG uygulaması geliştirdi. Bu uygulama ile müşterilerin beğendikleri eşyaları satın almadan kendi ev ortamlarında deneyimleyebildiği IKEA Place uygulamasından bir kesit Şekil I.4'te gösterilmektedir.



Şekil I.4 Ikea Place uygulaması

Öte yandan çeşitli teknoloji üreticileri hem sanal hem de AG deneyimini bir arada kullanılması amaçlanan Karma Gerçeklik (Mixed Reality) deneyimlerini sunmaya başladılar. Karma Gerçeklik (KG) ile sanal ve gerçek dünyanın bir araya getirilmesi sonucunda gerçek zamanlı ve etkileşimli bir deneyim sağlanmaktadır. Böylelikle sanal ve fiziksel dünyalar tek bir gerçeklik olarak sunulmaktadır.

KG ve AG farklı olarak sadece fiziksel ortam içerisine sanal objelerin veya verilerin eklenmesi değildir. Kullanıcılar gerçek zamanlı olarak fiziksel

mekânda bu sanal objeler ve veriler ile etkileşime geçebilmektedirler. Yine kullanıcılar, başka kullanıcılar ile birlikte yaşadıkları deneyimi etkileşimli olarak paylaşabilirler. KG için Microsoft'un Hololens'i ve çok ses getiren Magic Leap One gibi ürünleri örnek olarak gösterebiliriz.

Şekil I.5 Karma Gerçeklik donanımları

Dinamik bir yapıda olan zamanla birlikte kullanıcı alışkanlıkları ve teknoloji de değişiklik göstermektedir. Mobil cihazlara olan ilginin ve dolayısıyla mobil kullanımın yıldan yıla büyük bir hızla artış göstermesi, kullanıcıların ihtiyaçlarını istedikleri an, istedikleri yerde giderebilme ihtiyacından kaynaklanmaktadır. Bu ihtiyaçları karşılamak adına mobil donanım ve teknoloji üreticileri arasında oluşan rekabet mobil cihazların gelişimini çok hızlı bir şekilde etkilemiştir. Bu gelişmeler sonucunda yüksek performanslı olarak çalışan mobil cihazlar son kullanıcıların kullanımına sunulmuştur.

Mobil cihazların donanım gelişimi, yeni nesil bilgisayarlarda bulunan gelişme, çalışama, yürütme ortamına göre kısıtlı ve düşük seviyede de olsa Makine Öğrenmesi, Derin Öğrenme, Artırılmış Gerçeklik, Örüntü Tanıma ve Görüntü İşleme gibi yüksek seviyeli performans ihtiyacı olan teknolojilerin mobil cihazlarda yer almasına sebep olmuştur.

AG uygulamalarının mobil bir platformda olması hem kullanılabilirlik hem de ilgi çekiciliği bakımından bilgisayarla sunulmasından daha avantajlıdır. Mobil cihazlar üzerinde çalışan AG teknolojisi uygulamaları başta reklamcılık olmak üzere üretim sektörü, sağlık, eğitim, oyun ve benzeri alanlarda kullanılmaktadır. Mobil AG uygulamaları kullanıcıya sundukları iki ve üç boyutlu görsel destekleri, video oynatabilme, sanal objelerle sanal olarak etkileşim sağlama, konum tabanlı GPS bilgisi gibi özelliklerle zenginleştirilmiştir. Bu sayede algılanan gerçeklik kullanıcıya zenginleştirilerek sunulmaktadır.

AG uygulamalarında kullanıcı etkileşimi için kullanılan çeşitli yöntemler vardır. Bu yöntemlerden birisi hareket ve eylemleri tanıma işlemidir. AG'de bir sanal nesne ile insan arasındaki etkileşim ve bir nesneyi uygun bir şekilde manipüle etmek önemli bir konudur. AG'de geleneksel 2 boyutlu imge tabanlı tanıma ve etkileşim tekniği, kullanıcı ile sanal nesne arasında doğal bir etkileşim gerçekleştirmek için bir sınırlı bir yapıya sahiptir [4]. Bu sınırlı işlemlerde ise çeşitli donanım işaretleyiciler veya özel olarak tasarlanan gözlükler ve sensörler kullanılmaktadır.

Yapılan bu çalışmada Derin Öğrenme, Örüntü Tanıma, Görüntü İşleme gibi teknolojiler ayrı disiplinler olarak AG uygulamasında bir arada kullanılmıştır. Mobil AG oyunu ile sadece tek kamera kullanarak; sanal ortam üzerinde, bir fiziksel nesne ile bir sanal nesnenin fiziksel dünya kurallarının yer aldığı etkileşiminin sağlanması amaçlanmıştır.

Derin öğrenme ve derin Öğrenme Algoritmaları, Örüntü Tanıma ve Görüntü İşleme, AG teknolojileri farklı birçok alanda uygulanmıştır. Bu çalışmada kullanılan teknoloji çözümleri ayrıntılarıyla sonraki bölümlerde anlatılacaktır.

II. KULLANILAN TEKNOLOJİLER

2.1. Derin Öğrenme

Derin Öğrenme (Deep Learning), yapay zeka (Artificial Inteligence) kavramının içinde barındırdığı makine öğrenmesi (Machine Learning) konusunun özelleştirilmiş bir halidir.

Derin Öğrenme (DÖ) teknolojisi tasarlanırken, insan sinir sisteminin biyolojik yapısından ilham alınmıştır. Bu biyolojik yapıya ait görme, işitme, hareket etme, düşünme ve öğrenme yeteneklerinin matematiksel olarak tasarlanarak bilgisayar ortamına aktarılması ve bilgisayarın kendi kendine öğrenme süreçlerini sürdürülebilmesi amaçlanmıştır. Bu gereksinimler doğrultusunda hazırlanan yazılım ve donanım organizasyonları DÖ sistemlerini oluşturmaktadır.

Derin yapay sinir ağları için, klasik yapay sinir ağlarının çok katmanlı ve çok nöronlu özel bir hali olduğu söylenebilir [5].

Derin öğrenme temel olarak verinin temsilinden öğrenmeye dayalıdır. Bir görüntü için temsil denildiğinde; piksel başına yoğunluk değerlerinin bir vektörü veya kenar kümeleri, özel şekiller gibi özellikler düşünülebilir. Bu özelliklerin içinden bazıları veriyi daha iyi temsil etmektedir. Bu aşamada yine bir avantaj olarak, derin öğrenme yöntemleri, elle çıkarılan özellikler (handcrafted features) yerine veriyi en iyi temsil eden hiyerarşik özellik çıkarımı için etkin algoritmalar kullanmaktadır [6].

2.1.1. Evrişimli Sinir Ağları

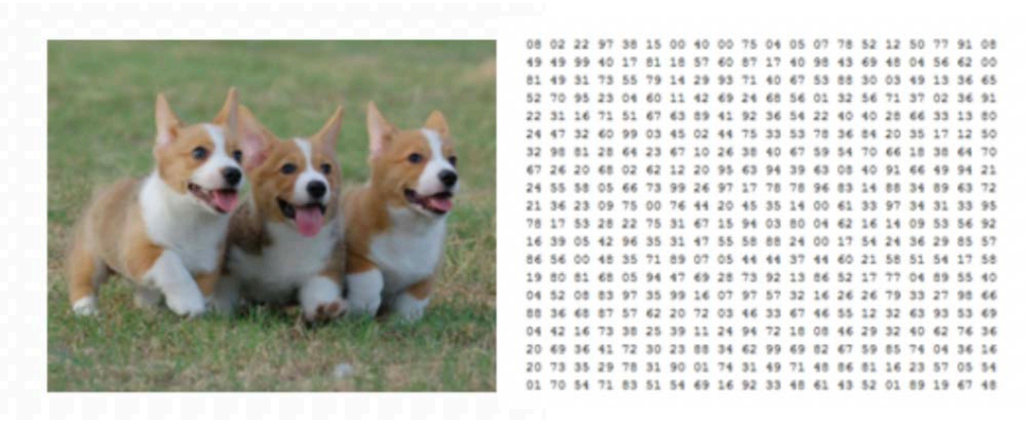
Diğer sinir ağları gibi bir Evrişimli Sinir Ağları (Convolution Neural Network) (CNN) bir giriş katmanından, bir çıkış katmanından ve aralarındaki birçok gizli katmandan oluşur. Evrişimli Sinir Ağları (ESA) ileri yönlü bir sinir ağı algoritmasına sahiptir.

ESA biyolojik görme korteksinden ilham almaktadır. Korteks'te bulunan küçük hücreler görsel alanın belirli bölgelerine duyarlıdır. 1960'larda Hubel ve Wiesel gerçekleştirdikleri bir deneyde; bazı bireysel çalışan nöronların görüntüler üzerinde sadece belirli bir yönelime ait çizgilere tepki verdiklerini tespit ettiler. Buna göre bazı nöronlar, yatay kenar çizgiler, bazı nöronlar ise dikey veya diyagonal kenar çizgiler gösterildiğinde tetiklenmektedirler. Yapılan bu çalışma ile; basit ve kompleks nöronların birlikte organize edilerek görsel algı üretebildikleri keşfedilmiş oldu. Görsel korteks bünyesinde özel karakteristik işlevler için çalışan nöron hücreleri fikri ESA'nın temelini oluşturdu [7].

Bir ESA, bir ya da daha fazla gizli katmanları içerir. Ancak tam bağlı bir sinir ağının aksine, ESA kendi girişleri üzerinde görüntü işleme gerçekleştiren seyrek olarak bağlı evrişimsel katmanlardan oluşan bir kombinasyon kullanır. Evrişim işlemi, matematiksel olarak bir nöronun kendi uyarı alanından uyaran nöronlara vermiş olduğu bir cevap olarak ifade edilebilir. Buna ek olarak nöron sayılarının azaltılması için katmanların birleştirmesi olarak adlandırılan ortaklama (pooling) katmanlarını da içerir. Genellikle ortaklama katmanını, çıkış katmanına bağlamak için bir ya da daha fazla tam bağlı katman (Full

Connected Layer) içerirler. ESA yapısında; girdi, evrişim, düzleştirilmiş doğrusal birim katmanı (RELU), ortaklama (pooling), sinir ağlarının bağlanması için tam bağlı katmanlar (Full Connected Layer) ve çıktı gibi birimler bulunur.

ESA'da ilk işlemi gerçekleştiren katman her zaman bir evrişim katmanıdır. Görüntü giriş olarak alınır. Bilgisayarın gördüğü görüntü verisi, o görüntüye ait piksel değerleri dizisi olacaktır. 224 x 224 ölçeğinde renkli bir görselimizin olduğunu varsayarsak; temsilci dizimiz 224 x 224 x 3 olacaktır. 3 değeri RGB değerlerini ifade eder. Oluşan matris içerisinde o piksel yoğunluğunu gösteren 0 – 255 arasında değerler atanır. Bilgisayar, görüntüler üzerinde bir sınıflandırma olasılığı oluşturabilmesi için bu sayı dizisine ihtiyaç duyar. Bilgisayardan beklediğimiz tanımasını istediğimiz nesnenin benzersiz özelliklerini bulmasıdır. Bir kedi görseline baktığımızda kedinin bıyıklarının olması, 4 ayağının olması, patileri ve benzeri özellikleriyle tanımlanabilmesiyle sınıflandırma gerçekleştirilebilir. Bilgisayarlarda, eğriler ve kenar çizgileri gibi soyut seviyede özellikler üzerinde arama yapar. Sonrasında bir dizi evrişim katmanı ile daha soyut kavramları temel alarak görüntü sınıflandırmasını gerçekleştirebilir. Şekil II.1'de insan ve bilgisayar görüsü arsındaki fark gösterilmektedir.

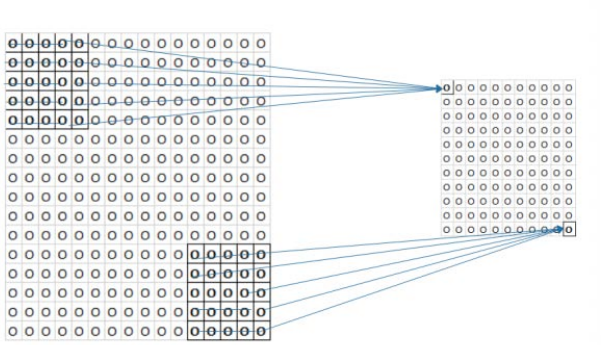


Şekil II.1 Bilgisayar Görüşü

Evrişim katmanındaki her nöron önceki katmandaki küçük bir nöron kümesinden sorumludur. Evrişim işleminde nöronların sayısını belirleyen sınırlayıcı kutu filtre veya hücre (cell) olarak da isimlendirilir.

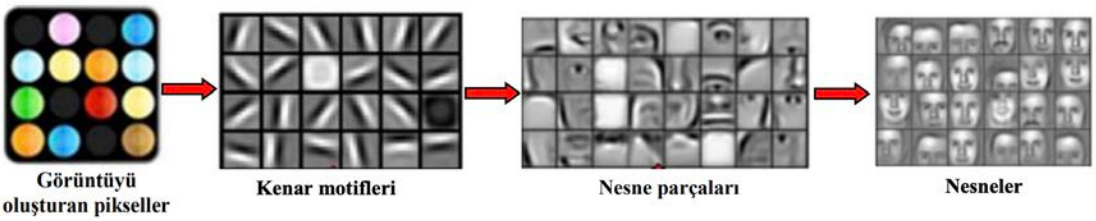
32 x 32 x 3 piksel sayı dizisinde bir giriş görüntüsü sağladığımızı varsayarsak; 5 x 5 bir filtrenin bu giriş görüntü üzerinde sol üst köşesinden başlayarak gezdiğini düşünebiliriz. Filtre, giriş görüntüsü üzerinde adım adım gezerek, görüntüdeki piksel değerleri, filtredeki değerler ile çarpar ve bu çarpım değerlerini toplar. Bu durumda sayı dizimiz için 75 çarpım (5 x 5 x 3) gerçekleştirmiş olur. Bu işlemi her bir konum için tekrarlar. Her benzersiz konum için bir sayı üretir ve kaydeder. Filtre boyutu 5 x 5 olan 32 x 32 giriş görüntüsü için 784 farklı konum elde edileceğinden, 32 x 32 x 3 piksel sayı dizisinden geriye 28 x 28 x 1 sayı dizsi elde edilir ve çıktılar bu diziye eşlenir. 5 x 5 x 3 iki adet filtre kullanıldığında çıkış değeri 28 x 28 x 2 olur. Evrişim işlemlerinde daha fazla filtre kullanarak detaylı tarama gerçekleştirilebilir. Daha sonra sonucu evrişim katmanındaki ilgili nörona aktarır [8]. Temsili

olarak Şekil II.2'te filtreleme işlemi ile özellik haritasının çıkartılması gösterilmektedir.



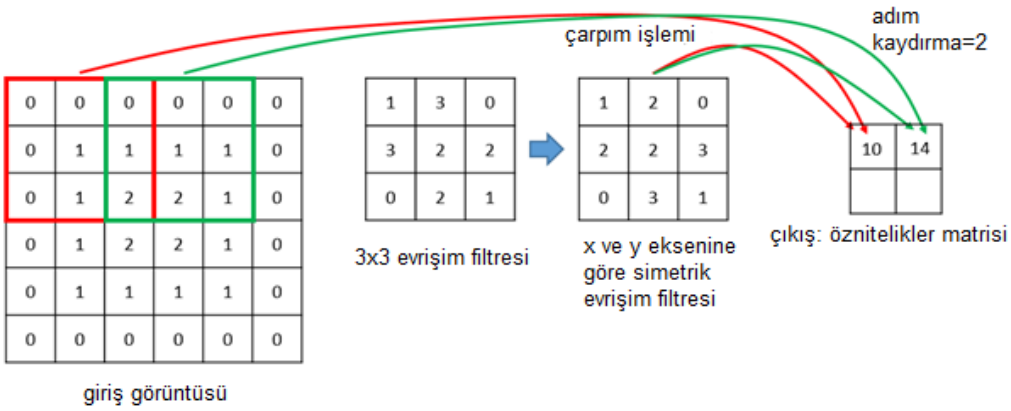
Şekil II.2 Filtreleme ile özellik haritasının çıkartılması

Filtreler, görüntüleri bulanıklaştırabilir, döndürebilir, keskinleştirebilir, kenarlıkları algılayabilir ve daha fazlasını yapabilir. Şekil II.3'te ESA'nın çeşitli katmanlarında gerçekleşen filtreleme işlemleri sonucu nesnelerin farklı temsilleri gösterilmektedir.



Şekil II.3 ESA'ların farklı katmanlarda gerçekleşen filtreleme işlemleri [9]

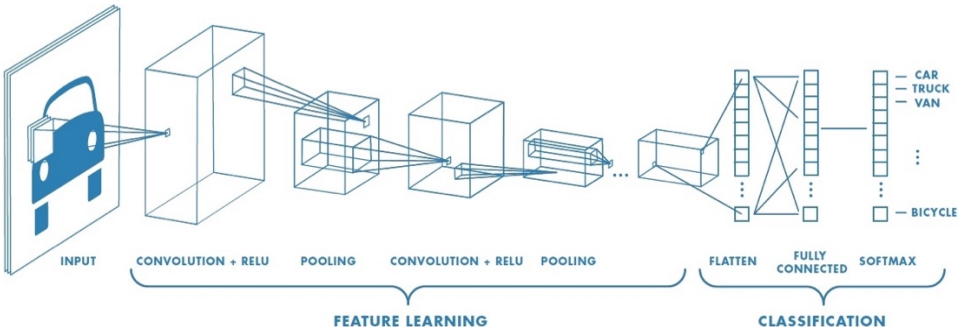
Özetle filtreler; görüntü üzerinde adım uzunluğuna bağlı olarak gezdirilirken her adımda çakışan değerler çarpılır ve sonuçlar üretilen çıktı matrisin elemanı olarak kaydedilir. Filtrenin simetriğinin alınmadığı durumda yapılan işleme ise korelasyon (cross corelation) denilmektedir. Şekil II.4'te bu filtreleme işlemleri gösterilmektedir.



Şekil II.4 Evrişim İşlemi ve filtreleme işlemleri

Tüm bu teknikler bir araya getirildiğinde tam bağlı bir sinir ağından farklı derin bir sinir ağı için mimari elde edilir.

ESA'da görüntü bir dizi evrişim işleminden, doğrusal olmayan pooling (ortaklama) ve tamamen birbirine bağlı katmanlardan geçirilerek bir çıktı elde edilir. Şekil II.5'te bir evrişimsel sinir ağının soldan sağa ilerleyişi gösterilmektedir.



Şekil II.5 Evrişimli Sinir Ağı modeli ilerleyişi

Görselde ortaklama (pooling) üzerinde çeşitli evrişim işlemleri gerçekleştirilir. Son özellik haritası daha sonra gizli katmanlara sığacak şekilde düzleştirilir. Daha sonra softmax fonksiyonu ile sınıflandırılır [10].

ESA'nın eğitim sürecinde sinir ağına etiketlenmiş binlerce resim verilerek, görüntünün ne olduğu öğretilir. ESA'da test verileri ile derin öğrenme algoritmasının öğrenme işleminde ne kadar başarılı olduğu test edilmektedir. ESA'lar, görüntüler üzerinde başlık eklemede, işlemede, sınıflandırmada, nesnelerin bölgesini belirlemede, ses, metin ve video işlemede yaygın olarak kullanılmaktadır [11].

2.1.1.1. Evrişimli Sinir Ağları Mimarileri

ESA mimarileri ve açıklamaları Tablo II.1'de açıklanmıştır.

Tablo II.1 ESA Mimarileri

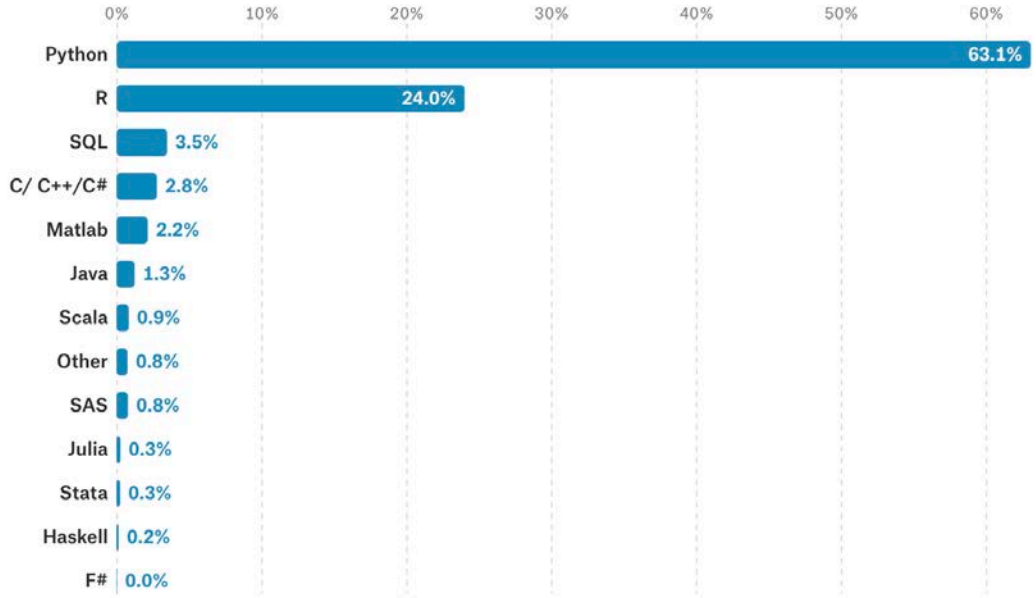
ESA Mimarileri	Açıklama
LeNet-5 [12]	1998 yılında LeCun tarafından ortaya çıkan 7 katmanlı öncü bir ESA modelidir. 32x32 gri tonlamalı el yazısı rakamlarını tanımak için kullanılmıştır. Yüksek çözünürlüklü görüntüler üzerinde işlem yapma yeteneği için yapısal olarak sınırlıdır.
AlexNet [13]	2012 yılında AlexNet, Alex Krizhevsky, Geoffrey Hinton ve Ilya Sutskever'den oluşan SuperVision grubu tarafından tasarlanmıştır.
ZFNet [14]	2013 yılında ILSVRC yarışması galibi bir CNN algoritması olan ZFNet oldu.
GoogLeNet/Inception [15]	Google'ın geliştirmiş olduğu GoogLeNet 2014 yılında ILSVRC yarışması galibi oldu. Adında ithaf edildiği üzere LeNet'ten ilham alınan bir CNN yapısını temel almıştır.
VGGNet [16]	2014 yılında ILSVRC yarışmasında ikinci olmuştur. VGGNet ağırlık yapılandırması geliştirmesi konusunda açıktır, özelleştirilebilir yapısı vardır.

ResNet [17]	ILSVRC yarışmasında 2015'te Kaiming He ve ekibi tarafından ortaya çıktı. Açılımı Residual Neural Network (Artık Sinir Ağı) olarak adlandırılmaktadır.
DarkNet [18]	C ve CUDA tabanlı olarak geliştirilen açık kaynaklı ESA algoritmasıdır. CPU ve GPU desteklemektedir.

2.1.2. Derin Öğrenmede Yazılım ve Donanım Gereksinimleri

Derin öğrenme modellerinin 2010'lu yıllarda gelişmesinin en önemli sebebi donanımsal kısıtların aşılmış olmasıdır. 90'lı yıllarda bilgisayarlarda yeterli işlemci gücü olmadığından, tasarlanmış olan ağ modellerinden verimli sonuçlar alınamamıştır. Günümüzde grafik işlemci birimleri (GPU) sayesinde özellikle vektör ve matris işlemleri çok daha hızlı yapılabilmektedir. Böylece karmaşık derin öğrenme ağ modellerinin de uygulanabilirliği sağlanmaktadır. Bununla birlikte dünyaca bilinen büyük servis sağlayıcıları bulut (cloud) servislerini de son kullanıcıya ücretli/ücretsiz şekillerde açmaktadır. Böylece yüksek kapasiteli işlem yapabilen makinelere uzaktan bağlantı sağlanarak derin öğrenme modelleri tasarlayıp eğitmek mümkün olmaktadır.

Makine öğrenmesi ve derin öğrenme konularında kullanımlarına göre programlama dilleri istatistikleri Şekil II.6'da ki gibi gösterilmektedir [19].



Şekil II.6 Derin Öğrenmede tercih edilen programlama dili istatistikleri

Derin öğrenme çözümleri üretebilmek için geliştirilmiş çeşitli kütüphaneler bulunmaktadır. Bu kütüphaneler ve özellikleri Tablo II.2.'de açıklanmıştır.

Tablo II.2 Derin Öğrenme Kütüphaneleri ve Özellikleri

Kütüphane	Geliştirici	Programlama Dili	Özellikleri
TensorFlow [20]	Google	Python	-Hızlı derleme yapabilmektedir. -TensorBoard ile görselleştirme yapabilmektedir. -Veri ve model paralelliği sağlar. -GPU veya CPU'da paralel çalışabilmektedir.

Caffe [21]	Berkeley Vision and Learning Center (BVLC)	Python	-İleri beslemeli ağlar ve görüntü işleme konularında hızlıdır. -Hassas ayarlama (finetuning) için önceden eğitilmiş modelleri vardır. -Hiçbir kod yazmadan model eğitilebilir. -Python ara yüzü oldukça kullanışlıdır. GPU desteği vardır
Caffe2 [22]	Facebook	Python	-Python API ile C++ desteği sağlar. Berkeley yazılım dağıtım lisansı vardır. -GPU desteği vardır.
Torch/PyTorch [23]	Google/Facebook	Lua/ Python	-Birçok modüler parçayı birleştirmek kolaydır. -Yeni katmanları yazıp GPU üzerinde çalıştırması kolaydır. -Çokça önceden eğitilmiş model vardır.
Keras [24]	Francois Chollet-Google	Python	-Torch kütüphanesinden esinlenilmiş sezgisel bir API'dir. -Theano, TensorFlow, Deeplearning4j ve CNTK arka planda kullanmaktadır. -Hızlı büyüyen bir yapısı vardır. -GPU veya CPU'da paralel çalışabilmektedir.

MxNet [25]	Pedro Domingos, Amazon AWS	R, Python ve Julia	-Çoklu GPU desteği vardır. -Eğitim hızı ve verimliliği yüksektir. -Yeni katmanlar eklemek kolaydır.
CNTK [26]	Microsoft	C++	Geri planda Python API kullanımına izin verir.
DeepLearning4j [27]	Adam Gibson	Java, Scala	-Java sanal makinesi (Java Virtual Envoirement-JVM) tabanlı olmasıdır.
KNet [28]	Deniz Yüret, Koç Üniversitesi	Julia	Kolay anlaşılır olması ve ifade gücü yüksektir. GPU Desteği vardır
Theano [29]	Montreal Institute for Learning Algorithms (MILA) Lab.	Python	Eylül 2017'de resmi olarak Theano kütüphanesinin geliştirilmeye devam edilmeyeceği duyuruldu.
TuriCreate [30]	Apple	Python	- Apple 2016 yılında Turi yapay zeka & makine öğrenmesi teknolojileri sağlayan firmayı satın aldı. -TuriCreate, özel makine öğrenimi modellerinin geliştirilmesini kolaylaştırır. -Nesne algılaması, resim sınıflandırma, regresyon, kümeleme, yazı sınıflandırma gibi işleri yapılabilen bir kütüphanedir.

2.1.3. Mobil Uygulamalarda Derin Öğrenme Teknolojileri

Apple, Google, Facebook, Amazon ve Netflix gibi büyük şirketler akıllı öneri sistemlerini mobil uygulamalarına ve mobil alt yapı sağlayan platformlarına dahil etmiş durumdalar. Büyük miktarlarda kullanıcıya sahip olan bu teknoloji devleri, kullanıcılarından büyük çapta veri elde etmektedirler. Bu şirketler, makine öğrenimi ve derin öğrenme teknolojileriyle kullanıcılarına ve işletmelere değer katmak için sahip oldukları verileri anlamlı hale getirip işlemektedirler.

Siri, Google'ın yapay zeka sesli asistanı gibi mobil cihazlar üzerinde yerleşik çözümlerde yaygınlığını arttırmaktadır. Mobil uygulamalar üzerinden elde edilen veriler çok daha kişiselleştirilmiş yapıda olduğundan bu verilerin kullanımı özellikle sayısal pazarlama sektöründe ciddi bir uygulama alanı oluşturmaktadır.

Mobil cihazlarda derin öğrenme ve yapay zeka teknolojileri, yenilikçi ürün üretmek ve çeşitli yeni fikirler elde etmek için şirketler tarafından tercih edilen, yatırım yapılan bir teknoloji olarak karşımıza çıkmaktadır.

2.2. Artırılmış Gerçeklik

AG, fiziksel dünyamızı genişleten, üzerinde sayısal bilgi katmanlarını ekleyen bir teknolojidir. Sanal Gerçeklik'ten (Virtual Reality) (VR) farklı olarak AG, gerçek olanı sanal olanla değiştirmek için tümüyle yapay ortamlar oluşturmaz. AG, Sanal Gerçekliğin (SG) aksine; kullanıcılar, bilgisayarlı görü tarafından geliştirilen gerçek dünyanın farkında olurlar. AG mevcut bir çevrenin doğrudan görünümünde görünür ve o fiziksel ortama

ortamdan aktarılan görüntülere sesler, videolar, grafikler, boyutlu nesneler ve GPS konum bilgisi ekler.

AG, fiziksel bir gerçek dünya ortamı üzerine zenginleştirilmiş içerikler ve görüntülerle gerçeklik algısını değiştirir [31]. Günümüzde AG, internet ve akıllı telefon teknolojilerinin gelişmesiyle çoğunlukla eş zamanlı etkileşimli konsepti ile karşımıza çıkmaktadır.

AG, ekranlar, akıllı gözlükler, el tipi görüntü aygıtları, akıllı telefonlar, başa takılan ekranlar ve robotik gibi çeşitli cihazlarda uygulanmaktadır. Bu gibi cihazlarda örüntü tanıma (Object Recognition) ve SLAM (Simultaneous Localization And Mapping) teknolojisi sayesinde, kamera görüntüsünden eş zamanlı haritalama, yerleştirme ve konumlandırma yapabilmektedir. SLAM, bilgisayarlı görüntü teknolojilerinde kullanılan, fiziksel dünyadan görsel verileri makineye ilişkin bir anlayış oluşturmak için 2 boyutlu özellik noktaları kullanan bir teknolojidir. Tespit edilen 2 boyutlu özellik noktaları arasında ilişkilendirme yaparak cihazın 3 boyutlu konumunu ve yönünü küçük ölçeklerde hesaplayabilir [32]. Şekil II.7'de görsel üzerinde tespit edilen SLAM 2 boyutlu özellik noktaları gösterilmektedir.

Şekil II.7 SLAM 2 boyutlu özellik noktaları

2.2.1 Artırılmış Gerçeklik Kütüphaneleri

Kişisel bilgisayarlar, akıllı telefonlar, tabletler ve akıllı gözlükler üzerinde artırılmış gerçeklik çözümleri üretebilmek için çeşitli kütüphaneler bulunmaktadır. Bu kütüphanelere ait detaylar Tablo II.3.'te gösterilmiştir.

Tablo II.3 Artırılmış Gerçeklik Kütüphaneleri

Paket	Platform
ARToolKit [1]	Android, iOS, Linux, Windows, Akıllı Gözlükler platformlarını destekler. Ücretsiz ve açık kaynak kodlu bir kütüphanedir.
Wikitude [2]	Android, iOS, Windows, Akıllı Gözlükler platformlarını destekler. Deneme sürümü bulunur, kullanımı ücretlidir.
ARKit [33]	Apple'ın geliştirdiği iPhone ve iPad cihazları üzerinde AG uygulamaları ve oyunları geliştirme için sunulan bir SDK'dır.
ARCore [34]	Google'ın geliştirmiş olduğu iOS ve Android cihazlarda desteklenen AG platformudur.

Vuforia [35]	Android, iOS, UWP, Unity ortamlarını destekleyen ücretli bir AG platformudur.
DeepAR [36]	Mühendis, araştırmacı ve 3D tasrımcı ve animasyon üreticileri tarafından yapay zeka, derin öğrenme, bilgisayarlı görü teknolojileri ile ilgili yayın ve teknoloji üreten bir ekibin üretmiş olduğu iOS, Android, HTML5 ve Unity tabanlı platformları destekleyen SDK'dır. Deneme sürümleri ücretsiz olarak sunulmaktadır.

2.3. Kullanılan Programlama Dilleri

2.3.1. Python

Python programlama dili tez çalışmasında bardak, fincan, kupa vb. nesnelerin tespiti için veri ön ayıklama, veri işleme, neşe tespiti eğitimi ve test süreçlerinde kullanılmıştır.

Mac OS işletim sistemi üzerinde Python tümleşik dağıtıcısı olan Anaconda ile yapılandırılmıştır. Gerçekleşen işlemlerde Python 2.7 sürümü kullanılmıştır.

2.3.2. C++

Mobil kamera üzerinde gerçek zamanlı olarak nesne tespiti operasyonlarına paralel olarak bardak ağzı boyutu hesaplama işlemlerinde kullanılmıştır. IOS platformu üzerinde kullanılan C++ paketleri Tablo II.4.'te gösterilmiştir.

Tablo II.4 Kullanılan C++ Paketleri

Paket	Modül
OpenCV	<opencv2/imgcodecs.hpp> opencv2/highgui/highgui.hpp <opencv2/imgproc/imgproc.hpp> <opencv2/imgcodecs/ios.h> <opencv2/core/eigen.hpp>
Eigen	<Eigen/Dense>
math	math.h
Algorithm	algorithm

2.3.3. Objective C

IOS platformu üzerinde C++ geliştirmelerini, Swift programlama dili ile kullanabilmek için sarmalama (wrapper) yöntemi ile kullanılmıştır. Swift programlama dili üzerinden C++ geliştirmelerine erişebilmek adına köprüleme (bridging) yapılmıştır. Sarmalama yapılan C++ kodu için Objective C .m dosyası (implemantation file) Default – C++ Source olarak imzalanmıştır.

2.3.4. Swift

Geliştirme aşamasında Swift 4.2 versiyonu kullanılmıştır. Kullanılan swift paketleri Tablo II.5.'te açıklanmıştır.

Tablo II.5 Kullanılan Swift Paketleri

Paket	Modül
RxSwift	Arka planda çalışan zamanlanmış işlemleri düzenlemek ve yönetmek için kullanılmıştır.
Darwin	Bu paket Swift üzerinde matematiksel işlemleri bünyesinde toplayan açık kaynaklı modülüdür. Çoğu fonksiyonlar, işlemler C Programlama dilinden taşınmıştır.
CoreMotion	Cihaz içinde bulunan ivmeölçer gibi sensörlere bilgi sunar.
Each	Zamanlayıcı kütüphanesidir.
Async	Swift üzerinde asenkron işlemler için kullanılan kütüphanedir.

2.4. Kullanılan Kütüphane Çözümleri

2.4.1. TuriCreate

Apple makine öğrenmesi ve yapay zeka alanında çalışan Turi firmasını 2016 Ağustos'ta satın aldı ve 2017 Aralık'ta TuriCreate'i açık kaynak haline getirdi. TuriCreate kolay kullanıma sahiptir, en önemli özelliğini motto olarak dile getirmektedirler. "Özel makine öğrenimi modellerinin geliştirilmesini kolaylaştırır". Bu slogandan anlaşılacağı üzere geliştiricilerin sıfırdan bir makine öğrenimi model oluşturması ve bu modeli kendileri özgü verilerle yapılandırmaları çok zaman alan bir sürece sahip olmasına karşın,

TuriCreate kütüphanesi bu işlemi nispeten diğer kütüphanelere göre hızlandırmaktadır.

TuriCreate, Nesne Sınıflandırma için “ResNet” veya “SqueezeNet” gibi ESA modelleri ile uyumlu bir şekilde çalışmaktadır. Aynı zamanda Nesne Tespiti “YOLOv2” algoritmasını kullanarak nesne tespitini oldukça performanslı bir şekilde gerçekleştirebilmektedir. Ayrıca detaylı ve gelişmiş seviyede Derin Öğrenme ve Makine Öğrenimi süreçlerinin yürütülmesi için TuriCreate “MxNet” kütüphanesi ile uyumlu bir şekilde çalışabilmektedir [37].

Bu çalışmada mobil kamerası üzerinden gerçek zamanlı olarak tespit edeceğimiz bardak, fincan, kupa vb. görüntülerin nesne tespitini yapmak adına TuriCreate kütüphanesinin “object_detection” modülü kullanılmıştır. Bu çalışmada hazırlanan öğrenmiş veri modeline ait çalışmalar IV. Sistem Gerçeklemesi bölümünde detaylıca anlatılmaktadır.

2.4.5. Vision

Vision yazılım kütüphanesi, iOS, macOS, tvOS sistemleri üzerinde yüz ve yüz simgesi tespiti, metin tespiti, barkod tanıma, görüntü kaydı ve genel özellik takibini gerçekleştirir [38].

2.4.2. CoreML

CoreML, Apple’ın 2017 yılında duyurduğu; Kamera, Siri ve QuickType’ın da dahil olmak üzere Apple ürünlerinde kullanılan yeni bir Makine Öğrenim yazılım kütüphanesidir. CoreML ile, eğitimli makine

öğrenme modellerini iOS, macOS, tvOS, watchOS sistemleri üzerinde çalışacak uygulamalarda entegre edilmektedir [39].

Şekil II.8 CoreML model seviyesi

2.4.3. ARKit

ARKit, bir AR deneyimi oluşturma görevini kolaylaştırmak için cihaz hareket izleme, kamera sahnesi yakalama, gelişmiş sahne işleme ve ekran kolaylıklarını bir araya getiren yazılım kütüphanesidir. En yaygın AR deneyimi türleri, iOS cihazının diğer görsel içeriklerle desteklenmiş arka kamerasından bir görünüm sergiler ve kullanıcıya çevrelerindeki dünyayı görmesi ve onlarla etkileşime geçmesi için yeni bir yol sunar [33].

2.4.4. OpenCV

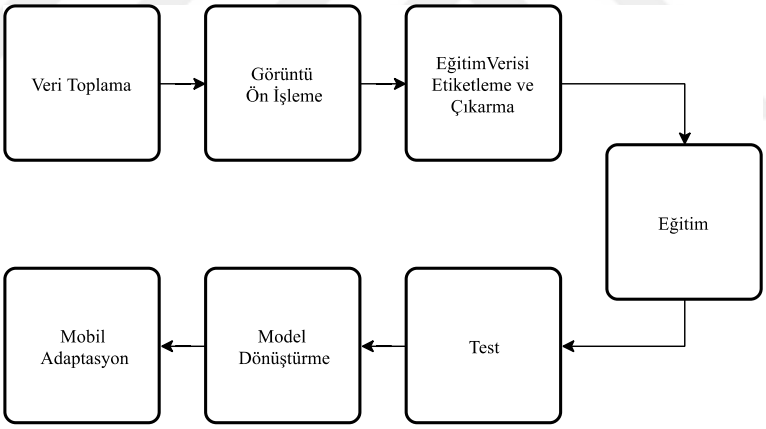
IOS platformu için uyarlanmış OpenCV2 3.2.0 Contrib kütüphanesi kullanılmıştır. Kullanımı sırasında yerel olan C++ dili ile geliştirmeler yapılmıştır [40].

III. SİSTEM TASARIMI

Derin Öğrenme Algoritması Kullanan Bir Mobil Abartılmış Gerçeklik Oyunu Sistem Tasarımı; hazırlık evresi ve kullanıcı ile etkileşim evresi olmak üzere iki aşamadan oluşmaktadır. Aşağıdaki paragrafta bunlar açıklanmaktadır.

3.1. Hazırlık Evresi Tasarımı

Hazırlık evresi, mobil uygulama öncesi işlemleri olarak derin öğrenme süreçlerini kapsar. Şekil III.1'de hazırlık evresi aşamaları gösterilmektedir. Bu tasarımdaki her bir aşama IV. bölümde detaylıca açıklanmıştır.

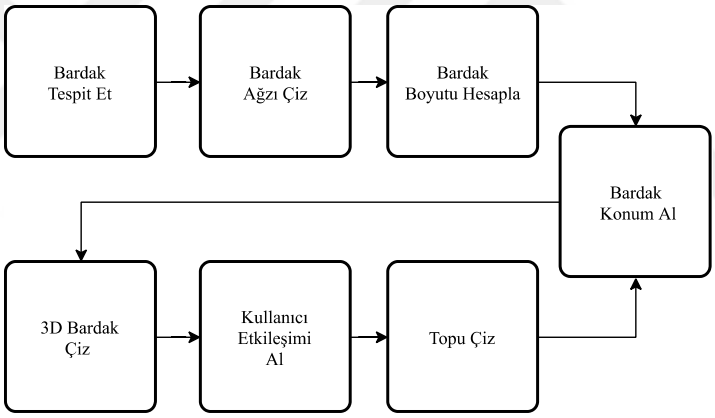


Şekil III.1 Hazırlık evresi tasarımı

3.2. Kullanıcı ile Etkileşim Evresi Tasarımı

Kullanıcı ile etkileşim işlemleri, oyunun başlaması ve oyun oynama evrelerinden oluşmaktadır. Öncelikle oyunun başlaması için gerekli olan süreçler kullanıcıdan eylemsel olarak talep edilmektedir. Kullanıcıdan beklenen oyun hazırlığı süreçlerinin tamamlanması ile birlikte oyun oynama

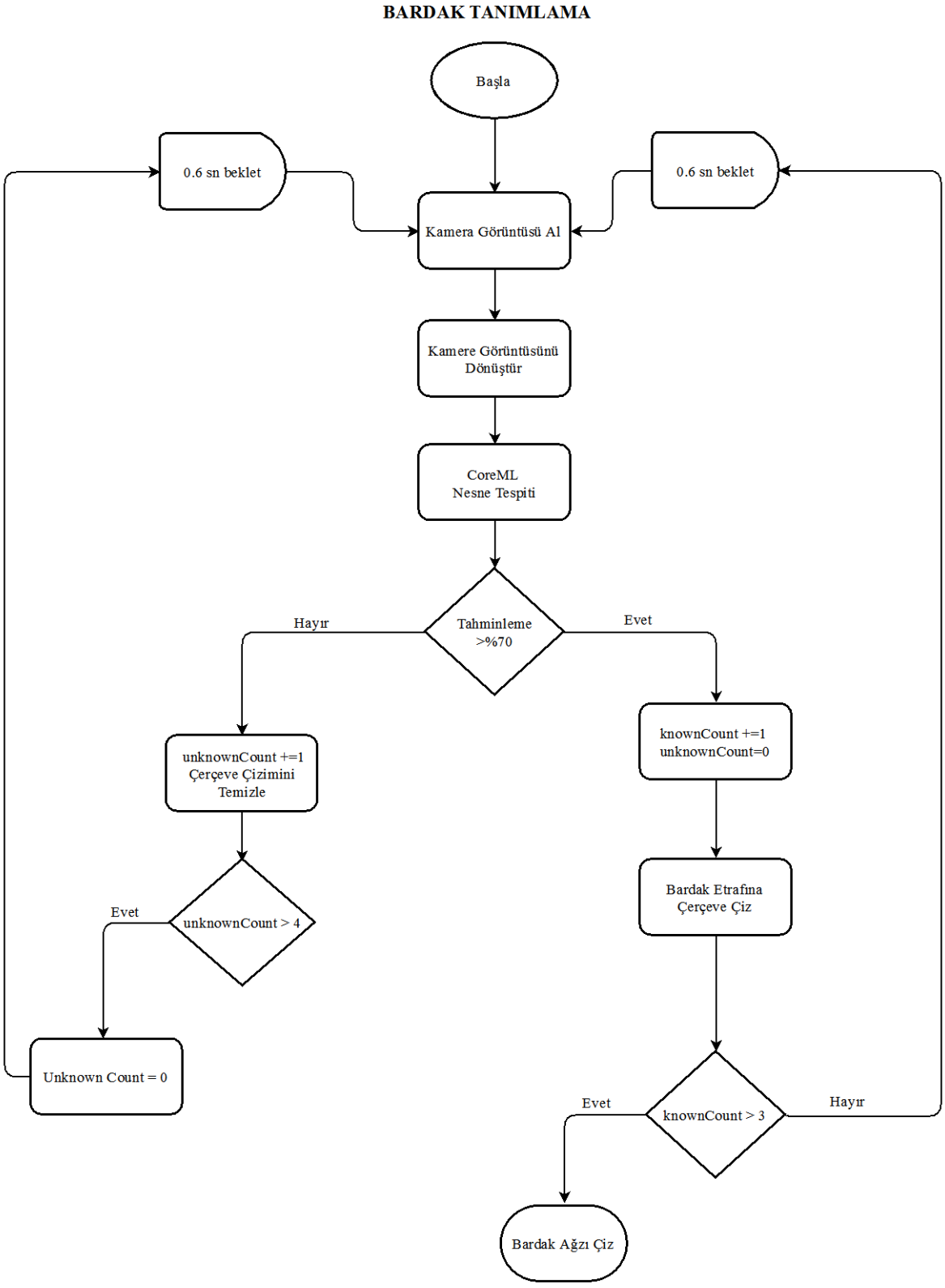
evresi başlamaktadır. Oyun oynama sürecinde, kullanıcı eylemleri ve talepleri geliştirilen oyun yazılımı tarafından toplanır. Kullanıcı yönlendirmeleri sonucunda belirlerken kamera bakışı, topun yönü, şiddeti gibi değişkenlerle topun fırlatılması sağlanmaktadır. Tespit edilen bardağın konumunda ve boyutunda 3 boyutlu olarak yerleştirilen sanal tüp ile fırlatılan top fiziksel olarak etkileşime geçer ve çeşitli geri bildirimler kullanıcıya sunulur. Etkileşim evreleri; Bardak Tanımlama, Bardak Ağzı Belirleme, Bardak Boyutu ve Konum Belirleme, Oynama Evresinden oluşmaktadır. Şekil III.2'de Kullanıcı ile etkileşim evreleri gösterilmektedir.



Şekil III.2 Kullanıcı ile etkileşim tasarımı

3.2.1. Bardak Tanımlama Tasarımı

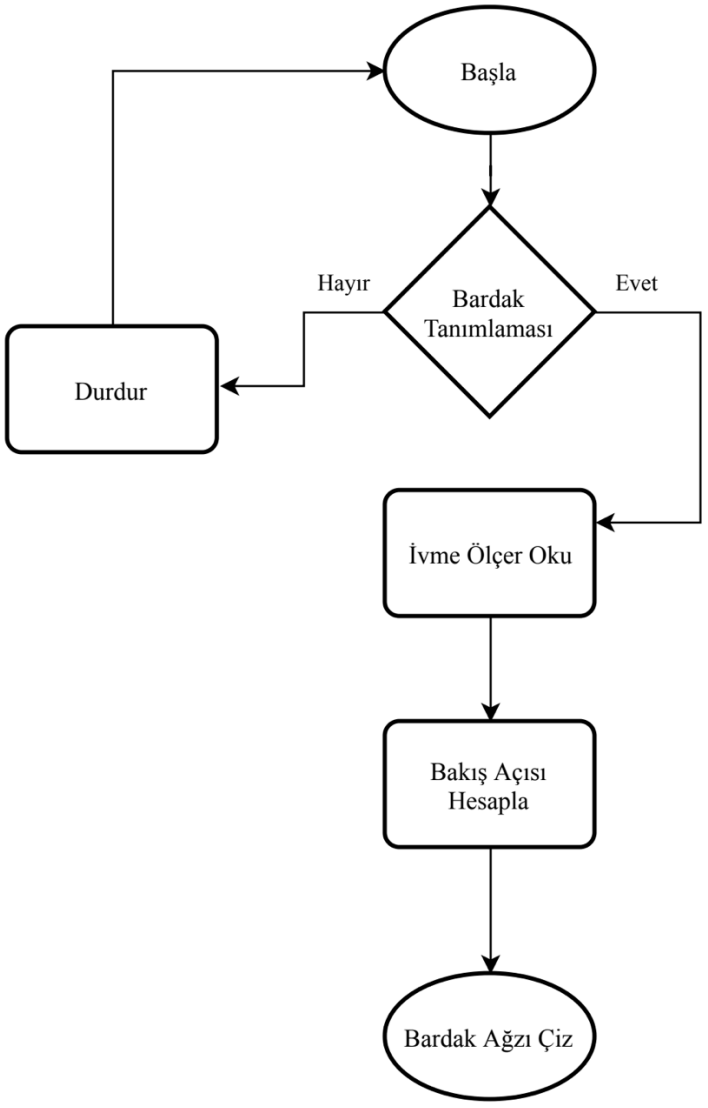
Gerçek zamanlı olarak mobil kamera üzerinde derin öğrenme ile desteklenen bu aşamada Bardak Tanımlama tasarımı Şekil III.3'te gösterilmiştir.



Şekil III.3 Bardak tanımlama tasarımı

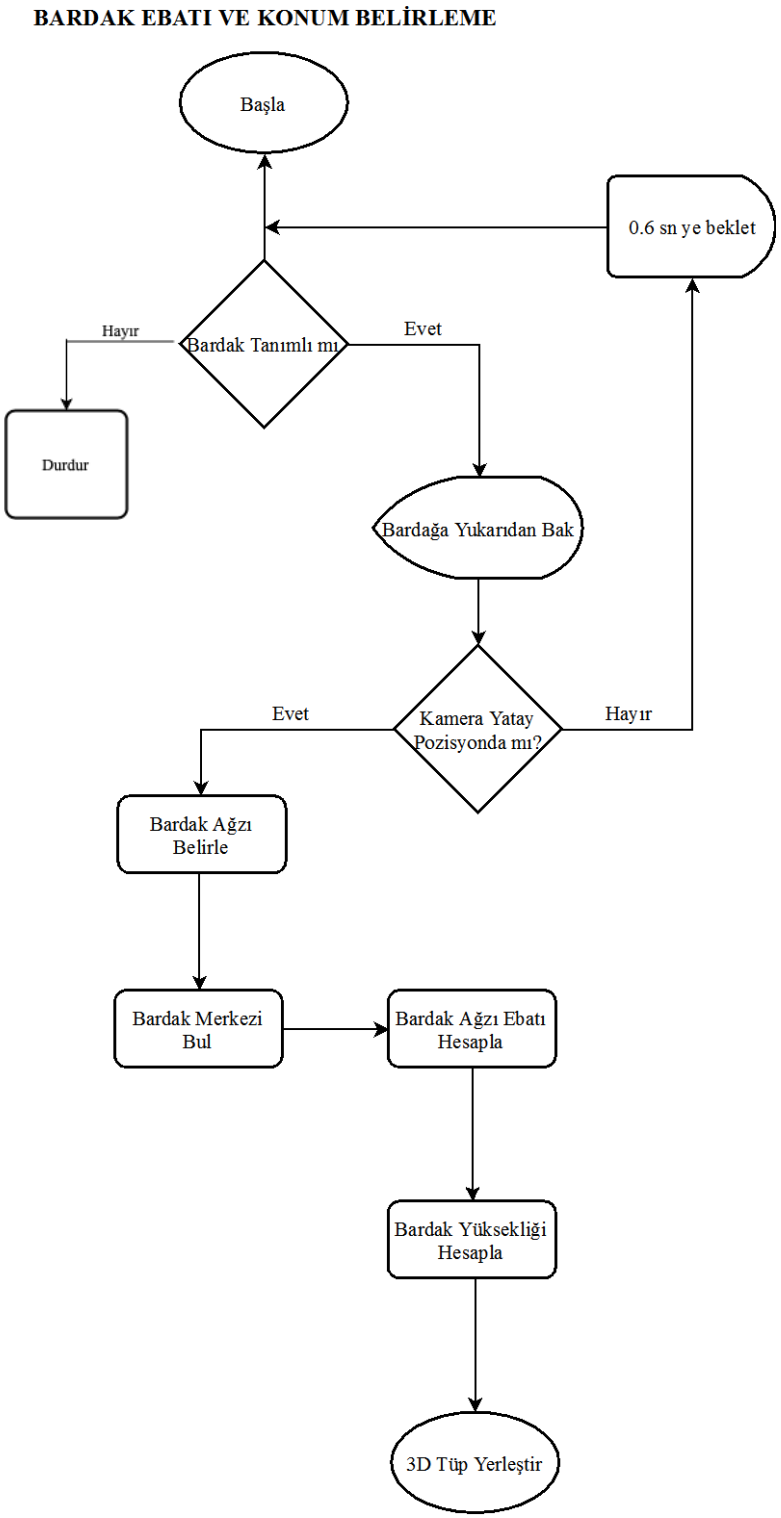
3.2.2. Bardak Ağızı Belirleme Tasarımı

Kullanıcının Bardak Tanımlama Evresini tamamlaması sonrasında Bardak Ağzı Belirleme evresi otomatik olarak başlatılır. Algoritma Şekil III.4'te gösterilmektedir.



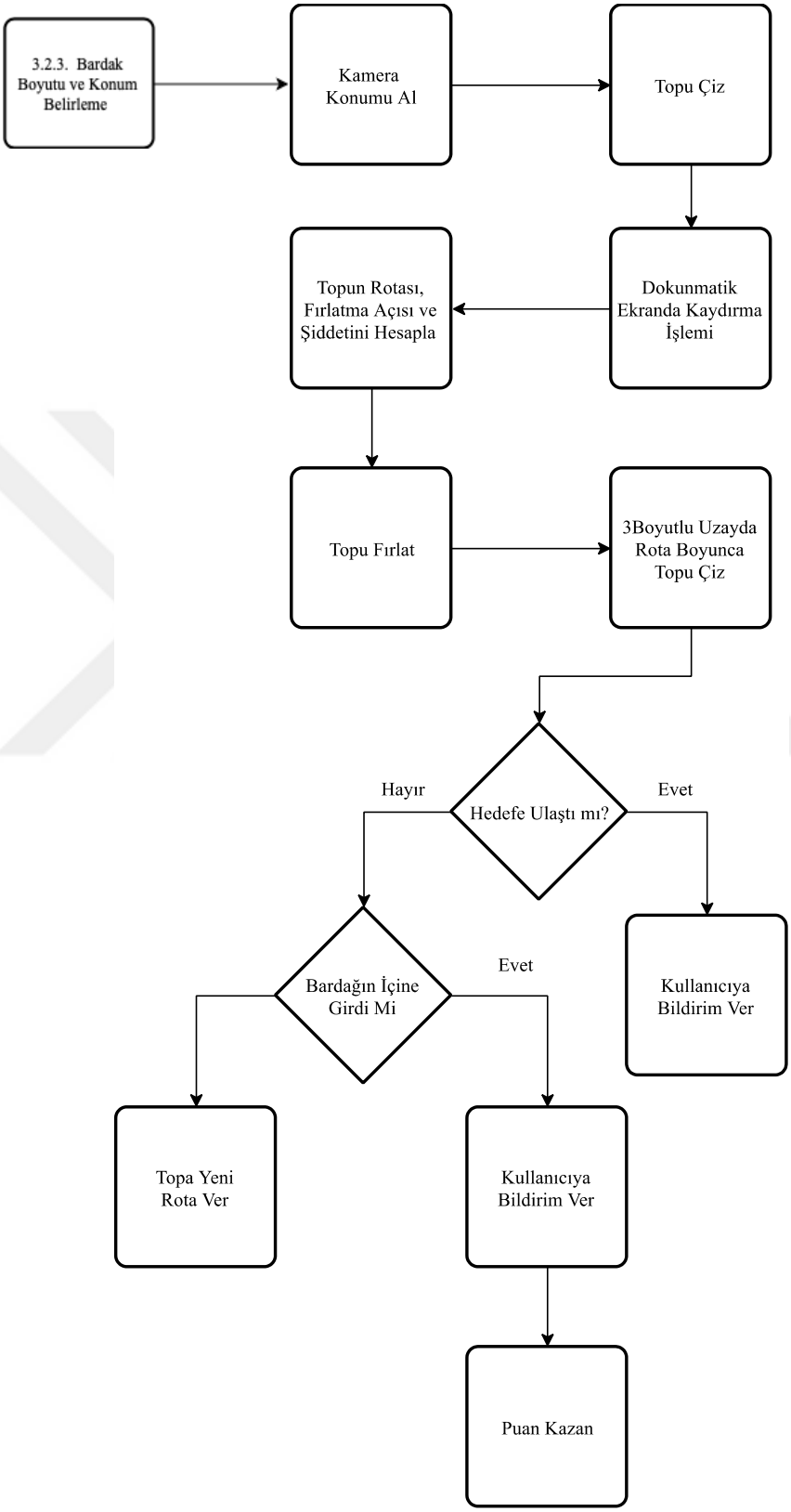
Şekil III.4 Bardak ağızı belirleme evresi tasarımı

3.2.3. Bardak Boyutu ve Konum Belirleme Tasarımı



Şekil III.5 Bardak boyutu ve konum belirleme evresi tasarımı

3.2.4. Oyun Gerçekleme Tasarımı

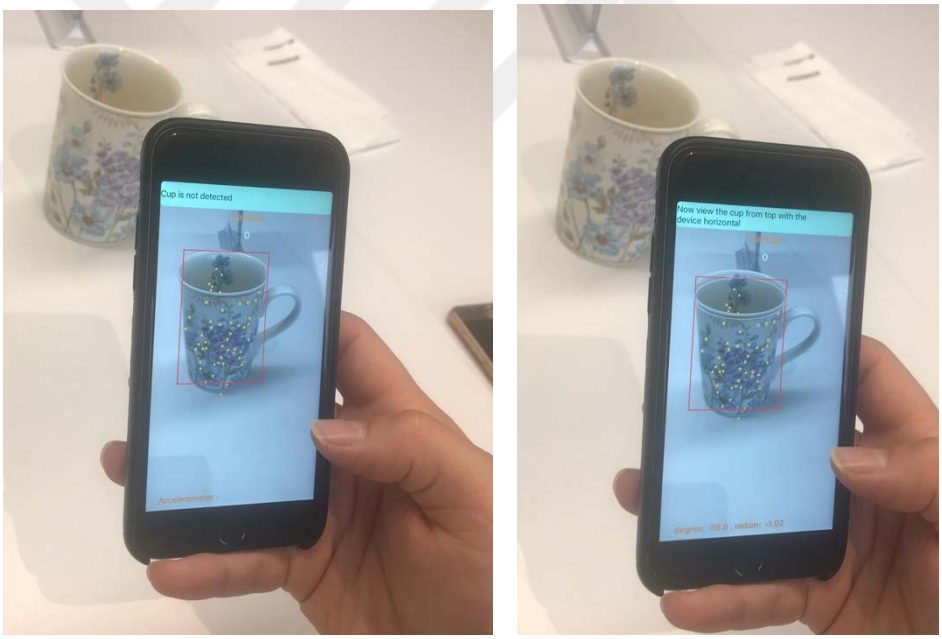


Şekil III.6 Oyun gerçekleme tasarımı

3.3. Oyunun Oynanması

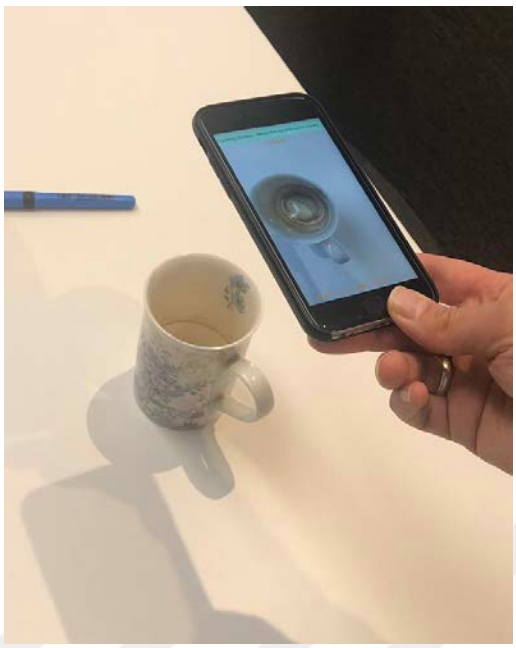
Bu çalışma; oyuncunun, mobil kamera üzerinden algıladığı gerçek dünyada bulunan bardağın içine sanal bir topu atması ve topun, aldığı ivme sonucu, bardak ile etkileşime geçmesinin betimlendiği bir artırılmış gerçeklik oyunudur.

Derin öğrenme algoritması ile mobil kamera tarafından öğrenilmiş model ile bardak tespiti yapılması sonrasında bardağın ağzı ve boyutlandırılması için hesaplamalar gerçekleştirilir. Şekil III.7'de bardak tanınması ve bardak ağızı çizilmesi gösterilmektedir.



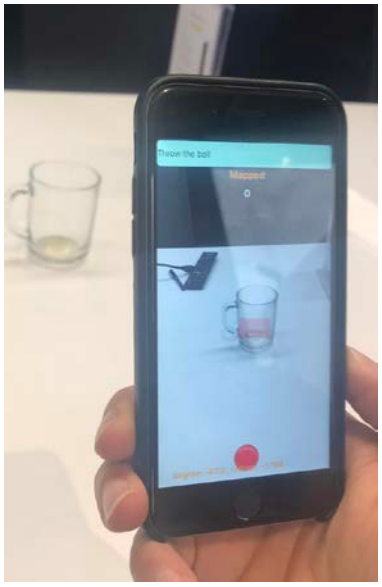
Şekil III.7 Oyun özeti bardak tespiti

Sanal uzayda bardak konumu hesaplanarak, oyun için gerekli olan etkileşime geçilecek bardak yerine bir silindirik nesne konumlandırılır. Şekil III.8'de bardak konumunun ve bardak boyutunun belirlenmesi gösterilmektedir.



Şekil III.8 Oyun özeti bardak konumu ve boyutu

Kullanıcı dokunmatik ekran üzerinden kaydırma işlemi yapar. Topun rotası, topun şiddeti ve açısı hesaplanır. Topun, 3 boyutlu uzayda etkileşimi sonucunda hedef bardağın içine girip girmediği tespit edilir. Oyun bu döngü içinde tekrarlanır. Şekil III.9'da oyunun başlangıcı gösterilmektedir.



Şekil III.9 Oyun özeti topun gösterimi

IV. SİSTEM GERÇEKLEMESİ

4.1. Hazırlık Evresi Gerçeklemesi

Veri Toplama, Görüntü Ön İşleme, Eğitim Verisi Etiketleme ve Çıkarma, Eğitim, Test, Model Dönüştürme, Mobil Adaptasyon aşamalarının gerçeklemesinde kullanılan yöntem ve çalışmalar aktarılacaktır.

4.1.1. Veri Toplama Gerçeklemesi

Mobil cihaz kamerası üzerinden gerçek zamanlı nesne tespiti için çalışmada kullanılacak kupa ve bardak nesnelerine ait görsellerin temin edilme aşamasıdır. Bu çalışmada bardak, fincan, kupa vb. nesnelerin temini için IMAGENET veri tabanı kullanılmıştır. IMAGENET Örüntü Tanıma, Örüntü İşleme gibi teknolojilerin araştırma ve geliştirme çalışmalarında görsel veri tabanı olarak kullanılmaktadır [41].

Aşağıda Şekil 4.1.'de temsili olarak gösterildiği gibi görsellerin bağlantıları txt uzantılı bir dosya içinde temin edilmiştir.

Şekil IV.1 ImageNet görsel temini

İndirilen dosya içinde yeni satırla ayrılmış 1000'in üzerinde kupa, fincan, bardak görsellerinin bağlantıları Şekil IV.2'de gösterilmiştir.

cup_urls.txt 〉 No Selection

```
http://www.allgifts.nl/utils/imaging/base/imaging.base.util.php?size=75&productID=324738
http://farm4.static.flickr.com/3213/2845504219_4cd956bcf2.jpg
http://farm3.static.flickr.com/2075/2971529654_57761bd11b.jpg
http://farm3.static.flickr.com/2019/2367538027_f4850110d3.jpg
http://farm3.static.flickr.com/2093/2167018453_28071ef738.jpg
```

Şekil IV.2 Görsel bağlantıları

İndirilen dosya içinde yeni satırla ayrılmış 1000'in üzerinde kupa, fincan, bardak görsellerinin bağlantıları Şekil IV.2'de gösterilmiştir. Bu bağlantılar Python programlama dili ile yazılmış bir kod bloğu ile mevcut dizine http bağlantısı ile otomatik olarak indirilmiştir. Bu kod bloğunda indirilebilecek sağlıklı bağlantılar üzerinden, dosya uzantısı alınarak dosya kendi formatında kayıt işlemi tamamlanmıştır. Bu işlemleri gerçekleştiren kod bloğu Şekil IV.3.'te gösterilmiştir.

```python
import requests
import os
import urllib2

array = []
with open("cup_urls.txt", "r") as f:
  for url in f:
    array.append(url)
    print(url)
    #last = url.split('/').pop().strip()
    last = url.rsplit('/', 1)[1]
    print(last)
    try:
        resp = urllib2.urlopen(url)
    with open(last.strip(), 'wb') as fl:
        fl.write(resp.read())
    except urllib2.HTTPError as e:
        print('HTTPError = ' + str(e.code))
    except urllib2.URLError as e:
        print('URLError = ' + str(e.reason))
    except Exception as e:
        print('generic exception: ' + str(e))
```

Şekil IV.3 Görsel bağlantılarını indiren Python kod bloğu

4.1.2. Görüntü Ön İşleme Gerçeklemesi

İndirilen görsel dosyaları, derin öğrenme algoritmasıyla nesne belirlemek için henüz hazır değildir. Nesne belirleme ve nesne sınıflandırma eğitiminin gerçeklenmesi için bu görseller üzerinde veri ayıklama ve düzenleme işlemleri gerçekleştirilecektir. Veri toplama ile elimizde bir görsel kütüphanesi oluşmuştur. Bu görsel kütüphane öğeleri üzerinde çeşitli işlemler gerçekleştirebilmek için yeniden bir düzenleme ile dosya adlandırma yapılır. Bu işlem kodları Şekil IV.4'te gösterilmektedir.

```
import os
count = 1
path = "/cup_images/"
for filename in os.listdir("cup_images"):
    print count
    count += 1
    name = "cup" + str(count) + '.' + os.path.splitext(filename)[1][1:].strip().lower()
    print filename, name
    os.rename(path+filename, path+name)
```

Şekil IV.4 Görselleri yeniden adlandıran Python kod bloğu

Mevcut görseller en ve boy olarak farklı farklı ölçeklere sahiptir. Görsel normalizasyon işlemini yapıp ya da yapmamak kullanılan ve tercih edilen ESA modellerine göre farklılık gösterebilir. Bunun için alternatif yöntemler bulunmaktadır. Bu yöntemlerden biri ise, görselleri toplu olarak yeniden boyutlandırmada yeni bir genişlik elde etmek için standart bir yükseklik yüzdesi seçilmesidir. Sonra bu görseller bir döngü içerisinde yeniden boyutlandırılıp bir klasöre kaydedilir. Yeni standart yükseklik değerini en boy oranıyla çarparak yeniden boyutlandırma için genişlik değeri elde edilir. Python 2.7 ve OpenCV kütüphanesi ile orijinal yüksekliğin yarısını standart

yükseklik olarak kullanan biçimlendirilmemiş bir örnek Şekil IV.5'te gösterilmiştir.

```
import cv2, os
folder = 'kaynak_dizin'
outfolder = 'hedef_dizin'
images = os.listdir(folder)
for i in images:
    fname = "\\".join([folder, i])
    img = cv2.imread(fname)
    H, W = img.shape[:2]
    aspect = W/(H*1.0)
    h = H * 0.5
    w = h * aspect
    resized = cv2.resize(img, (int(w), int(h)))
    cv2.imwrite('\\'.join([outfolder,i]), resized)
```

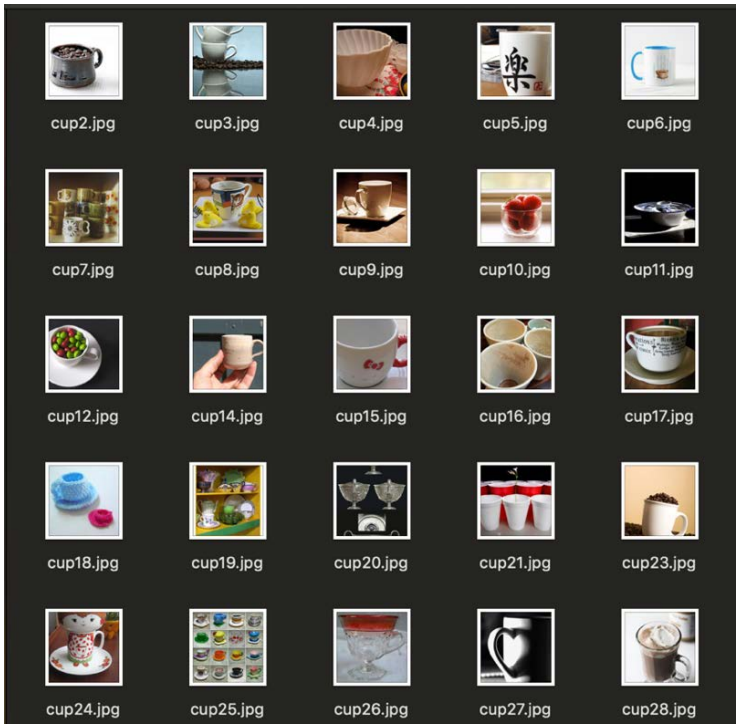
Şekil IV.5 Görselleri yeniden boyutlandıran Python kod bloğu

720 x 720 pikselli bir görsel için bir satırda 518.400 gibi vektör oluşmaktadır. Bu yapay sinir ağı içinde katmanlar (nöron sayısı) arttıkça buna bağlı olarak satır sayısı x 518.400 gibi bir matris oluşacaktır. Eğitim ve doğrulama esnasında daha hızlı sonuç almak için küçük vektörlerle işlem yapmak tercih edilen bir yöntemdir. Görselleri 224x224 çözünürlüğüne düşüren ve bozulmadan yeniden boyutlandıran kod parçası, Şekil IV.6'da gösterilmektedir.

```python
from PIL import Image
import os
from resizeimage import resizeimage
from pprint import pprint

path = "/cup_images/"
resize_path = "cup_resizeimage/"
image_list = os.listdir(path)
image_list.sort()

for img_file in image_list:
    pprint(img_file)
    with open(path+img_file, 'r+b') as f:
        with Image.open(f) as image:
            cover = resizeimage.resize_cover(image, [224, 224])
            cover.save(resize_path+img_file, image.format)
```

Şekil IV.6 224x224 olarak yeniden boyutlandıran Python kod bloğu

Öte yandan mobil cihaz kamerasından alınan gerçek zamanlı görsel verilerin tahminleme başarısının artması için 224x224 piksel oranında ölçeklendirilmiştir. Şekil IV.7'de bu ölçeklendirilmiş görseller gösterilmektedir.



Şekil IV.7 Yeniden adlandırılmış ve ölçeklendirilmiş görseller

4.1.3. Eğitim Verisi Etiketleme ve Çıkarma Gerçeklemesi

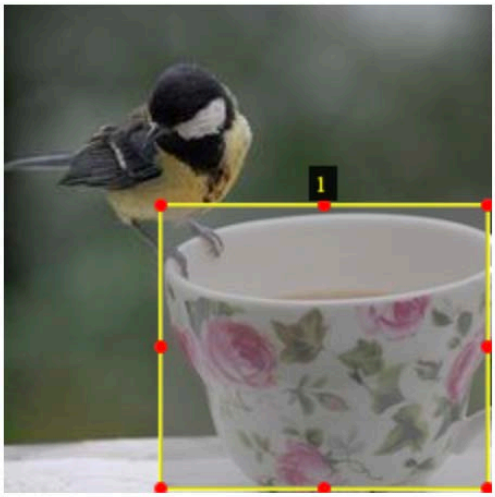
Ön ayıklamadan sonra görseller üzerinde etiketleme ve eğitim verisi çıkarma işlemleri yapılmalıdır. Ancak buradaki işlemler, önceki işlemlere göre daha fazla çaba sarf edilen bir aşama oldu. Bu aşama; görseller üzerinde bardak, kupa, fincan ve vb. nesnelerin algılanmasını ve algılanan nesnelerin görsel üzerindeki bölgesinin belirlemesini kapsar. Daha sonra hazırlanan bu veriler json uzantılı dosya içerisine konum (x, y) ve en-boy (width, height) olarak betimlenecektir. Görseller üzerinde bardak, fincan, kupa gibi nesnelerin bölge tespitini yapmada VGG Image Annotator (VIA) adlı araç kullanılmıştır. VIA, Oxford Üniversitesi Visual Geometry Grup tarafından görseller üzerinde bölgeleri tanımlamak ve bu bölgelerin metinsel tanımlarını oluşturmak için kullanılan bir görsel işaretleme aracıdır. Açık kaynaklı ve Html tabanlı bu araç bilgisayar üzerinde yerel olarak yüklenip kullanılmaktadır [42]. VIA aracı üzerinde gerçekleştirilen bölge tespiti çalışmalarına örnekler Şekil IV. 8'de gösterilmiştir.

Şekil IV.8 VIA aracı üzerinde yapılan bir bölge tespiti

Bir görsel üzerinde işaretlenen birden fazla nesneye ait bölge belirleme çalışması da Şekil IV.9'da gösterilmektedir.

Şekil IV.9 VIA aracı üzerinde birden fazla nesneye ait bölge belirleme

Bir görsel üzerindeki bölge belirleme sonrasında elde edilen json dosya çıktısı Şekil IV.10'da gösterilmektedir.




```
[
    {
        "label":"cup",
        "filename":"cup496.jpg",
        "coordinates":{
            "x":73,
            "y":93,
            "width":150,
            "height":130
        }
    }
],
```


Şekil IV.10 VIA aracı üzerinden json çıktısı

Bu çalışmada veri ayıklama işlemlerinden sonra elde edilen 645 adet görsel üzerinde bölge belirleme yapılarak bir eğitim veri kümesi elde edilmiştir.

4.1.4. Eğitimin Gerçeklenmesi

Nesne tespitinde eğitime verilecek görsellerin çeşitliliği üretilen eğitim modelinin başarısında etkileyecektir. Işığı az olan, başka nesne ile birlikte görüntülenen, farklı açılardan elde edilmiş ve betimlenmiş görseller çeşitliliği ne kadar fazla olursa, eğitim sonucunda elde edilen eğitim modeli kullanım esnasında aynı oranda başarı gösterecektir. Eğitim esnasında kullanılacak görseller çözünürlüğü ve dosya boyutu ne kadar düşük olursa eğitimin sonuçlanması da o kadar hızlı olacaktır. Eğitim ve test için görseller %80 ve %20 olarak iki gruba ayrılır. Grupların görsel çeşitliliği ne kadar çok olursa sistemin başarı oranı o kadar artar.

Tensorflow, Keras ve TuriCreate gibi kütüphaneler bu hazırladığımız eğitim veri kümesi yapılandırılmasında ve kullanılmasında ufak değişikliklere gereksinim duyabilir. Eğitim sonucunda oluşturulan öğrenilmiş Nesne Tespiti yapan modelimiz IOS cihazlarda uyumlu olarak çalışması için de bazı dönüşümlere gereksinim duymaktadır. Bu çalışmada dönüşümler için Apple'ın 2016 yılında bünyesine dahil ettiği TuriCreate kütüphanesi tercih edilmiştir [43].

Nesne tespiti eğitimi için kullanılan Python kütüphane paketleri Şekil IV.11'de gösterilmektedir.

```
import turicreate as tc
import os
import json
import ast
from pprint import pprint
from turicreate import SArrayBuilder
```

Şekil IV.11 Nesne Tespiti için Python paketleri

Nesne Tespiti için kullanılan görseller ve bu görseller üzerinde hazırlanan bölge tespitlerini de içeren json dosyası Python kodlarına bir değişken olarak tanımlanır. Kod satırlarında kullanıma dahil edilen TuriCreate kütüphanesi sonraki işlemlerde kısaltma olarak "tc" deyimi ile çağrılacaktır. SFrame sınıfı üzerinde bulunan "read_json" fonksiyonu ile json tipindeki csv uzantılı bir dosya veya json desenine uyan yerel ya da internet ortamı üzerinde bulunan json veri kümesi yüklenebilir. Bu çalışmada bölüm 4.1.3'te görsel kütüphanesi üzerinde işaretlenip hazırlanan json tipindeki veri kümesi yüklenir. Yüklenen bu json tipindeki Kod bloğunun devamında ise json dosya içinde TuriCreate kütüphanesiyle uyumlu çalışabilmesi adına, görsel veri kümesinde bulunan dosya adı ile json veri kümesindeki bölge tespitinde bulunan dosya adı düğümü sayesinde verilerin birbiriyle bağlama işlemini gerçekleştirilir. Bu işlemler Şekil IV.12 ve 13'te gösterilmektedir.

```
image_path = "cup_images/"
annotation_path = "converted_data2.json"
annotations = tc.SFrame.read_json(annotation_path, orient='lines')['X1'][0]
sb = SArrayBuilder(num_segments= 1, history_size = 10, dtype = tc.Image)
```

Şekil IV.12 .json içerisindeki X1 değeri kök düğümü

```
for annotation in annotations:
    json_data = json.loads(str(annotation).strip().replace("'",'"').replace('u"','"'))
    filename = image_path + json_data[0]["filename"]
    #print json_data[0]["filename"]
    sb.append(tc.Image(filename))
    #print filename
images = sb.close()
```

Şekil IV.13 Nesne Tespitinde görsel ile bölge tespiti verisini eşleştirme

Daha sonra görsellerimizi temsil eden "image", görseller üzerinde bölge tespitini yaptığımız veri listesini "annotation" olarak işaretleyip SFrame sınıfı nesnesinde tutulur. SFrame sınıfı ise, Pandas kütüphanesinde bulunan DataFrame sınıfından türetilmiştir. Ayrıca "annotation" içinde bulunan "filename" ile dosya adını "label" ile "cup" olarak imzaladığımız değerleri ayrı sütunlar olarak dışarı çıkartıyoruz. Sınıflandıracak nesne birden fazla olduğu durumda ise bu işlem mutlaka gerekli olacaktır. Mevcut görsellerin çeşitliliğini "cup" olarak sınıflandırdık. Her görsel içerisinde fincan, kupa, bardak hatta çay bardağı olarak sınıflandırmayı gerçekleştirebilirdik. Ancak, bu çalışmada bardak "cup" imzalanması için tüm çeşitliliğin "cup" olarak tanımlanması yeterli olur. Şekil IV.14'te bu işlemler gösterilmektedir.

```
data = tc.SFrame({"image": images, "annotations": annotations})

#path ayri olarak cikariyoruz
data["filename"] =data["annotations"].apply(lambda filename : [filename][0][0]["filename"])
data["label"] = data["annotations"].apply(lambda label : [label][0][0]["label"])
#print len(data["label"]), len(data["annotations"]),len(data["filename"]), len(data["image"])
```

Şekil IV.14 Nesne Tespitinde eğitim seti oluşturma

Eğitim için veri üzerinde %80 eğitim için %20 doğrulama verisi olarak ayrım yapılmıştır. TuriCreate paketi Nesne Tespiti modülünde

“object_detector” fonksiyonu üzerinden eğitim modeli oluşturulur. Öğrenilmiş “cup” nesne tespit modeli ve TuriCreate modeli (“cup_detection.model”) dosyası olarak kayıt edilir. Bu şekilde bardak nesnelerinin görsel üzerinde sınıflandırmasını ve görsel üzerindeki yerini belirleme işlemini yapacak modele ait eğitim kodları hazırlanmıştır. Şekil IV.15’te ise bu adımlar gösterilmektedir.

```
train_data, test_data = data.random_split(0.8)
# Create a model
model = tc.object_detector.create(train_data,feature='image',
                                  annotations='annotations',max_iterations=3000)
# Save predictions to an SArray
predictions = model.predict(test_data)
# Evaluate the model and save the results into a dictionary
metrics = model.evaluate(test_data)
# Save the model for later use in Turi Create
model.save('cup_detection.model')
```

Şekil IV.15 Nesne Tespitinde TuriCreate eğitim modeli oluşturma

Pyhon 2.7 ortamında hazırlanan kodların bulunduğu “cup_detection.py” dosyası çalıştırılır ve eğitim süreci başlatılmış olur. Eğitim sürecinde açık kaynaklı yapay sinir ağı kütüphanesi olan Darknet’in gerçek zamanlı olarak nesne tespiti gerçekleştiren YOLO (You Only Look Once) algoritmasını kullanır. Eğitim, “create” fonksiyonun eğitilmiş “ObjectDetector” modelinin döndürülmesiyle tamamlanır ve model değişkenine atama yapılır. Sonrasında test ve dönüşüm işlemleri için diğer kod satırları çalıştırılır. Kullanılan max_iteration değeri, görüntü başına gerçekleşecek eğitim yineleme sayısını ifade etmektedir. Bu çalışmada hazırlanan veri kümesi için 3000 yineleme sayısı “Loss” değerinin 0’ yaklaşması için yeterli olmuştur.

Eğitim, MacOS işletim sistemine sahip 2,7 GHz Intel Core i7 CPU'ya sahip bir bilgisayar üzerinde gerçekleşmiştir. Eğitim yaklaşık olarak 11 saat gibi bir sürede tamamlanmıştır.

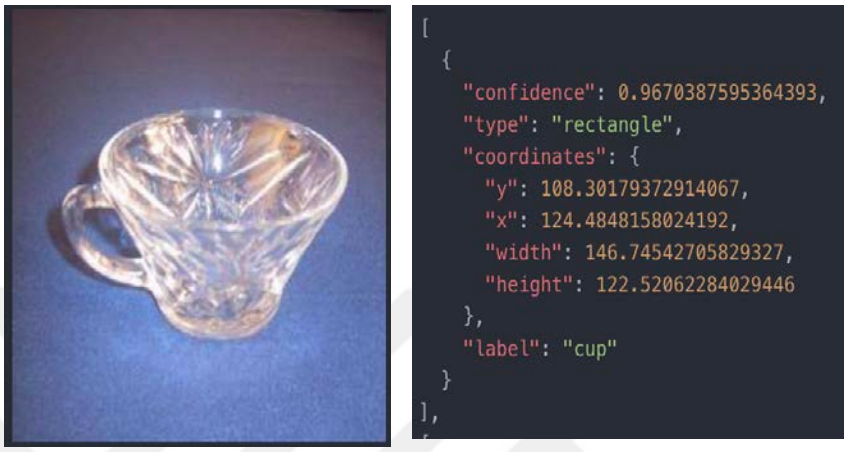
4.1.5. Test Gerçeklemesi

Eğitim verisi için kullanmadığımız kalan %20 içinde görsellerimiz ile eğitmiş olduğumuz Derin Öğrenme Nesne Tespiti modelimizi test işlemi gerçekleştiriyoruz. Bu aşamada eğitim modeli başarısı "confiedence" çıktısı 0-1 arasında yüzde olarak değerlendirilir. Öte yandan test görseli içindeki bölge tespitini ise test sonucunda "rectangle" düğümü olarak elde edilir. Şekil IV.16'da Derin Öğrenme Nesne tespiti test süreçleri gösterilmektedir.

```
import turicreate as tc
#load image
test = tc.SFrame({'image': [tc.Image('cup_images/cup544.jpeg'),
                            tc.Image('cup_images/cup585.jpg')]})
#load the model
model = tc.load_model('cup_detection.model')
test['predictions'] = model.predict(test)
print(test['predictions'])
#this part work only on Mac
test['image_with_predictions'] = \
    tc.object_detector.util.draw_bounding_boxes(test['image'],
                                                test['predictions'])
test.explore()
```

Şekil IV.16 Nesne Tespitinde TuriCreate test işlemi

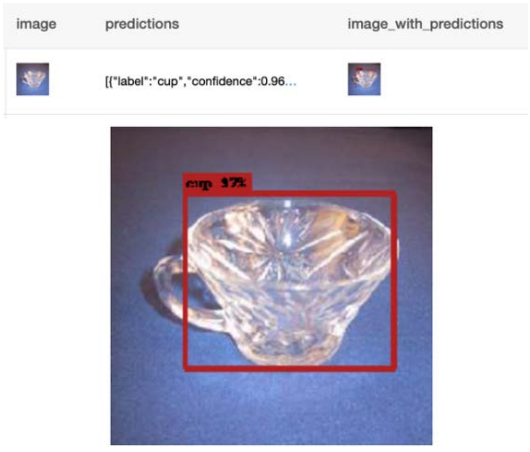
Yapılan test işlemi sonucunda başarı oranı ve bölge tespiti sonucu bir dizi olarak json tipinde çıktı alınmaktadır. Buna göre cup544.jpeg görseli ve bir test çıktısı Şekil IV.17'de gösterilmektedir.



Şekil IV.17 Nesne Tespitinde test çıktısı

Burada "confidence" çıktısı %96 başarılı bir bardak tanımlanmıştır.

TuriCreate üzerinde explore() fonksiyonu ile bölge tespiti test işlemi sonucunda çerçeve çizimiyle birlikte gösterilmektedir. Şekil IV.18'de bu çıktı gösterilmiştir.



Şekil IV.18 Nesne Tespitinde TuriCreate explorer() test çıktısı

Test etmiş olduğumuz görsel üzerinde birden fazla bardak nesnesi olması durumunda ise eğitilmiş olan modelimizin nasıl bir başarı göstermiş olduğunu gözlemlediğimizde ise %97'lik bir başarı oranı çıktısı almaktayız. TuriCreate üzerinde explore() fonksiyonu ile Şekil IV.19'da bu çıktı gösterilmiştir.

Şekil IV.19 TuriCreate ile birden fazla nesne tespiti test çıktısı

Çözünürlüğü eldeki veri setimize uymayan 1000x1250 boyutundaki bir görsel test işlemine girdiğinde %86'lık bir başarı ile sonuç alınmıştır.

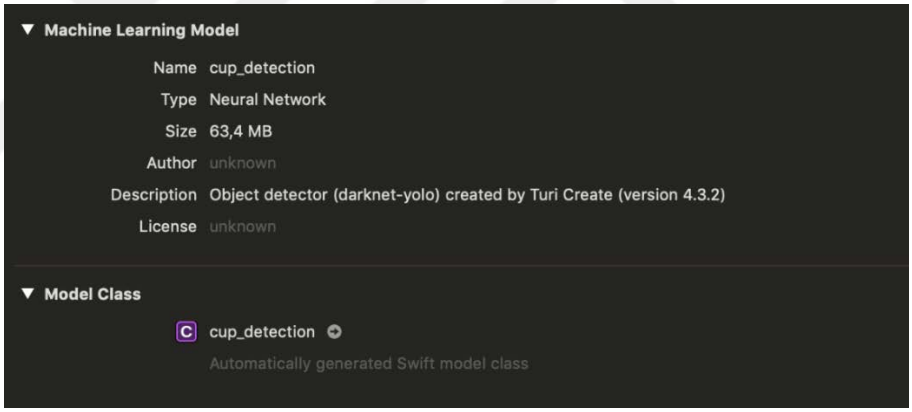
4.1.6. Mobil Adaptasyonu Gerçeklemesi

Bu çalışmanın yapıldığı sıralarda güncel olarak IOS Platformu üzerinde güncel olan Makine Öğrenim Kütüphanesi CoreML2'dir. Eğitmiş olduğumuz bir fotoğraf görseli içerisinde Derin Öğrenme Nesne Tespiti yaparak bardak tespitini bölge tespiti ile yapacak olan modelimizi IOS uygulamasında kullanabilmek adına CoreML2 üzerinde uyumlu çalışacak şekilde dönüştürmeliyiz. Bunun işlem Şekil IV.20'de gösterilmiştir.

```
# Save the model for later use in Turi Create
model.save('cup_detection.model')
# Export for use in Core ML
model.export_coreml('cup_detection.mlmodel')
```

Şekil IV.20 TuriCreate Nesne Tespiti modelin CoreML dönüşümü

Dönüştürülmüş “cup_detection.mlmodel” dosyası Arttırılmış gerçeklik uygulamasında kullanabilmesi için XCode geliştirme ortamına dosya olarak dahil edilmesi ve projenin derlenmesi gerekmektedir. XCode geliştirme ortamına dahil edilen “cup_detection.mlmodel” dosyasına ait künye Şekil IV.21’deki gibi gösterilmektedir.



Şekil IV.21 XCode üzerinde dahil edilen eğitim modeli

4.2. Kullanıcı ile Etkileşim Evresi Gerçeklemesi

Bu bölümde Bardak Tanımlama, Bardak Ağzı Belirleme, Bardak Boyutu ve Konum Belirleme, Oynama Evresi gibi kullanıcı etkileşim aşamaları açıklanacaktır.

4.2.1. Bardak Tanımlama Gerçeklemesi

Bardak Tanımlama gerçeklemesi için, uygulama ilk açıldığında kamera görüntüsü alma işlemleri için her 0.6 sn.'de bir tekrarlı olarak uygulama arka planında çalışan bir "thread" başlatılır. Bu tekrarlı işlemler içerisinde ARKit oturumundan bir kamera görüntüsü alınır, aygıt konumuna göre uyumlu bir şekilde 416 x 416 oranlarına orantılı olarak dönüştürülür. Derin Öğrenme Nesne Tespiti yapan "cup_detection.mlmodel" modeli kullanılarak CoreML kütüphanesiyle tahminleme gerçekleştirilir. Tahminleme başarısının %70 ve üstü olduğu durumda modelden alınan koordinatlar kullanılarak ekrana bir çerçeve çizilir. Öte yandan bardak bulma sayısı bir arttırılır. Kontrol amaçlı, ardışık kamera görüntüleri ile bardak bulma sayısı 3'ü geçtiği durumda sahne üzerinde bardak tanıma gerçekleşmiş olmaktadır. Eğer sahnede üst üste 4 adet bardak bulamama durumu oluşursa artık sahne değişmiş olarak algılanır ve yeniden bardak bulma işlemi için başa döndürülür. Şekil IV.22'de kamera görüntüsü alma süreçleri gösterilmektedir.

```
if !self.arCamInitializingIsOk{
    print("Not arCamInitializingNot OK")
    observer.onCompleted()
    return Disposables.create()
}
guard let frame = self.sceneView.session.currentFrame else {
    print("No frame available")
    observer.onCompleted()
    return Disposables.create()
}
// Create and rotate image
let pixbuff : CVPixelBuffer? = (frame.capturedImage)
if pixbuff == nil {
    print("NovarPixelBuffer available")
    observer.onCompleted()
    return Disposables.create()
}
```

Şekil IV.22 Swift ile kamera görüntüsü alma

CVPixelBuffer tipindeki varsayılan yüksek çözünürlük olarak aldığımız görüntü 416 x 416 çözünürlüğüne Şekil IV.23'te gösterildiği gibi dönüştürülmüştür.

```
let ciImage = CIImage(cvPixelBuffer: pixbuff!).rotate
let resizeBuffer = ciImage.convertUIImage().pixelBuffer(width:
    Int(self.targetImageSize.width), height: Int(self.targetImageSize.height))
```

Şekil IV.23 Swift ile görüntü çözünürlük dönüşümü

CIImage nesnesi üzerinde yazılan ek paket (Swift Extension) sayesinde "rotate" komutu ile cihaz oryantasyonuna (orientation) göre ölçeklendirme yapılmaktadır. Yine bu yöntem sağlanırken kamera oryantasyonuna göre cihaz konumu algılamak için ek paket yazılmıştır. Bu ek paketler Şekil IV. 24 ve 25'te gösterilmiştir.

```
var rotate: CIImage {
    get {
        return self.oriented(UIDevice.current.orientation.cameraOrientation())
    }
}
```

Şekil IV.24 Swift ile CIImage nesnesi üzerinde oryantasyon ek paketi

```
private extension UIDeviceOrientation {
    func cameraOrientation() -> CGImagePropertyOrientation {
        switch self {
        case .landscapeLeft: return .up
        case .landscapeRight: return .down
        case .portraitUpsideDown: return .left
        default: return .right
        }
    }
}
```

Şekil IV.25 Swift ile cihaz oryantasyonu alma ek paketi

CIImage sınıfı üzerinden çağrılan convertUIImage fonksiyonu aynı şekilde yazılmış bir ek pakettir. Bu işlem Şekil IV.26'da gösterilmektedir.

```
public func convertUIImage() -> UIImage
 {
     let context:CIContext = CIContext.init(options: nil)
     let cgImage:CGImage = context.createCGImage(self, from: self.extent)!
     let image:UIImage = UIImage.init(cgImage: cgImage)
     return image
 }
```

Şekil IV.26 Swift ile CIImage nesnesi üzerinde UIImage ek paketi

Dönüştürülen bu ekran görüntüsü CoreML kütüphanesi tarafından eğitmiş olduğumuz bardak modeli üzerinde tahmminleme gerçekleştirilecektir. Eğitilmiş modelimize ait kullanım bilgileri ve çıktıları ise Şekil IV.27.'de gösterilmiştir.

▼ Model Evaluation Parameters

Name	Type	Description
▼ Inputs		
image	Image (Color 416 x 416)	Input image
▼ Outputs		
confidence	MultiArray (Double 2535 x 1)	Boxes × Class confidence (see user-defined metadata "classes")
coordinates	MultiArray (Double 2535 x 4)	Boxes × [x, y, width, height] (relative to image size)

Şekil IV.27 Eğitmiş olduğumuz modelin kullanım parametreleri

XCode üzerinde Swift programlama dili ile bu model üzerinde işlem yapabilmesi için öncelikle import deyimi ile CoreML kütüphanesinde bulunan Vision paketi ekli olmalıdır. Daha sonrasında Swift üzerinde bu modeli değişken olarak kullanılması için de VNCoreModel sınıfı kullanılır. Şekil IV.28.'de bu kullanım gösterilmiştir.

```
/// - OBJECT DETECTION: MLModelSetup
let cupModel :  VNCoreMLModel = try! VNCoreMLModel(for: cup_detection().model)
```

Şekil IV.28 Eğitmiş olduğumuz modelin değişken ataması

Dönüştürülerek elde edilen görüntü Vision paketi içinde bulunan VNCoreMLRequest nesnesi ile nesne tespiti isteği başlatılmaktadır. VNImageRequestHandler nesnesi ile yapılan nesne tespiti isteğinin sonucu beklenmektedir.

```
let objectDetectionRequest = VNCoreMLRequest(model: self.cupModel, completionHandler: { request, error in
    guard error == nil else {
        print("ML request error: \(error!.localizedDescription)")
        observer.onCompleted()
        return
    }
    guard let observations = request.results as? [VNCoreMLFeatureValueObservation] else {
        fatalError("unexpected result type from VNCoreMLRequest")
    }
    observer.onNext([(observation: observations, image: ciImage, frame: frame, buffer: resizeBuffer!)])
    observer.onCompleted()
})
objectDetectionRequest.imageCropAndScaleOption = .scaleFit
do {
    let resizeNewCIImage = CIImage.init(cvPixelBuffer: resizeBuffer!)
    try VNImageRequestHandler(ciImage: resizeNewCIImage, options: [:]).perform([objectDetectionRequest])

} catch {
    print("ML request handler error: \(error.localizedDescription)")
    observer.onCompleted()
}
```

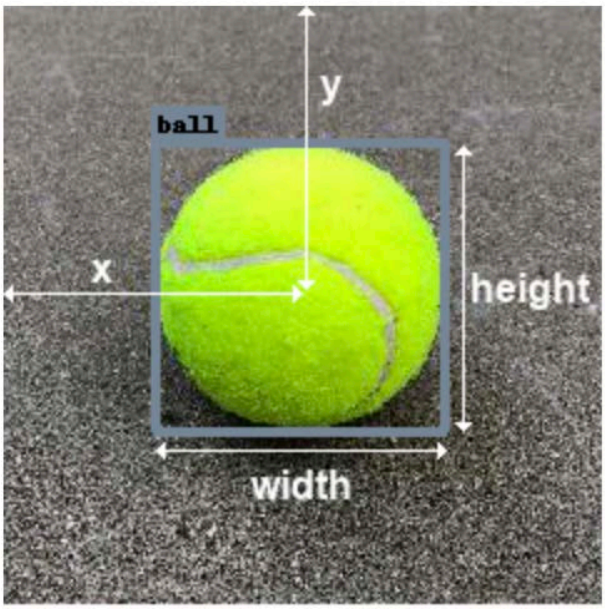
Şekil IV.29 Swift Nesne Tespiti isteği

Yapılan istek Gözlemci Tasarım Deseni (Observable Design Pattern) yapısındaki VNCoreMLFeatureValueObservation nesnesine atanmaktadır. Gelen yanıt çok boyutlu bir dizi (Multi Dimension Array) olarak temin edilmektedir. Oluşturulan Prediction sınıfı ile gelen çok boyutlu tahminleme yanıtı ayıklanmaktadır. Bu işlemin bir parçası Şekil IV.30.'da gösterilmektedir.

```
//Struct to handle multi arrays
class Predictions {
    static func FromMultiDimensionalArrays( observations: [VNCoreMLFeatureValueObservation]?,
        [Prediction]? {
        guard let results = observations else {
            return nil
        }

        let coordinates = results[0].featureValue.multiArrayValue!
        let confidence = results[1].featureValue.multiArrayValue!
```

Şekil IV.30 Swift Nesne tespiti çok boyutlu dizisi yanıtı ayıklama

Burada ki kritik nokta; TuriCreate kütüphanesinde eğitilen model üzerinden gelen bilgiler bir merkez ve bir çerçevedir. Halbuki bize gereken çerçeve sol üst köşe koordinatlarıdır. Bu ise hesaplanarak bulunur. Şekil IV 31.'de tahminleme yapılan nesnenin bölge belirlemesini, o nesnenin merkezine göre çerçeveye alınması gösterilmektedir.



Şekil IV.31 TuriCreate nesne tespiti bölge konumlandırması [43].

Çizilecek olan çerçevenin yükseklik, genişlik, merkezi x ve y değerlerine göre mobil cihazımızı uyumlu hale getirmek için Şekil IV.32.'deki hesaplama yapılmaktadır.

```
if maxConfidence > confidenceThreshold {
    let x = coordinatesPointer[b * 4]
    let y = coordinatesPointer[b * 4 + 1]
    let w = coordinatesPointer[b * 4 + 2]
    let h = coordinatesPointer[b * 4 + 3]

    let rect = CGRect(x: CGFloat(x - w/2), y: CGFloat(y - h/2),
                      width: CGFloat(w), height: CGFloat(h))

    let prediction = Prediction(labelIndex: maxIndex,
                                confidence: Float(maxConfidence),
                                boundingBox: rect)
    unorderedPredictions.append(prediction)
}
```

Şekil IV.32 Swift için bölge konumlandırması çerçevesi dönüşümü

Çok boyutlu dizi sonucuna sahip olduğumuzdan en başarılı tahminleme değeri kullanılmaktadır ve başarı oranına göre elde etmiş olduğumuz sonuç dizisi %70 ve üstü tahminleme başarısına göre sıralanmaktadır. Bu işlem Şekil IV.33'te gösterilmektedir.

```
let orderedPredictions = unorderedPredictions.sorted
{ $0.confidence > $1.confidence }.filter{$0.confidence > 0.70}
if orderedPredictions.count == 0 { return predictions}
```

Şekil IV.33 Swift tahminleme başarısı filtreleme

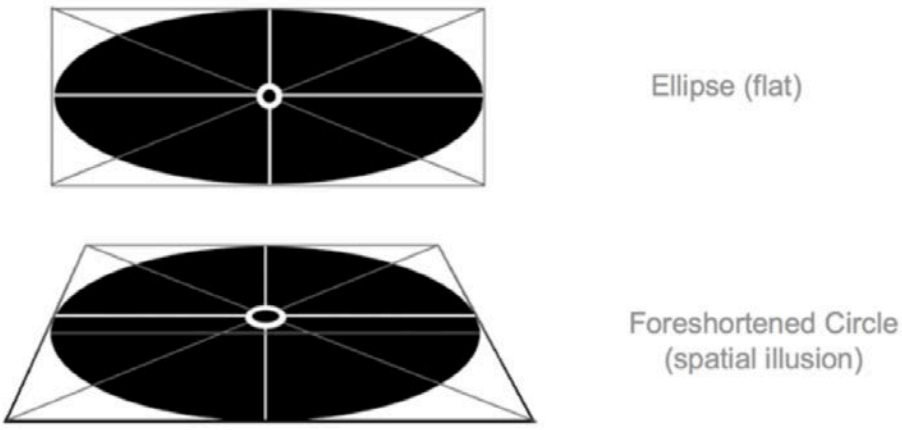
Gelen en başarılı tahmin üzerindeki boundingBox değeri ki bunu CGRect olarak dönüştürmüştük. Bu çerçeveyi ekran çözünürlüğüne uygun olarak sahneye çizme işlemimiz kalmıştır. Bu işlem için ise Şekil IV.34.'te gösterildiği gibi sahneye uygun şekilde çerçeve dönüştürülerek çizim gerçekleştirilmektedir.

```
func highlightArea(boundingRect: CGRect) {
    let source = self.sceneView.frame

    let rectWidth = source.size.width * boundingRect.size.width
    let rectHeight = source.size.height * boundingRect.size.height

    let compatibleRect = CGRect(x: boundingRect.origin.x * source.size.width,
                                y:boundingRect.origin.y * source.size.height,
                                width: rectWidth, height: rectHeight)
    DetectedRectangle.shared().setRectangle(rect: compatibleRect)
    let outline = UIView()
    //TAG 1 RECTANGLE
    outline.tag = 1
    outline.frame = compatibleRect
    outline.layer.frame = compatibleRect
    outline.layer.borderWidth = 2.0
    outline.layer.borderColor = UIColor.red.cgColor
    self.sceneView.addSubview(outline)
}
```

Şekil IV.34 Swift başarılı tahminleme çerçeve çizimi

4.2.2. Bardak Ağzı Belirleme Gerçeklemesi

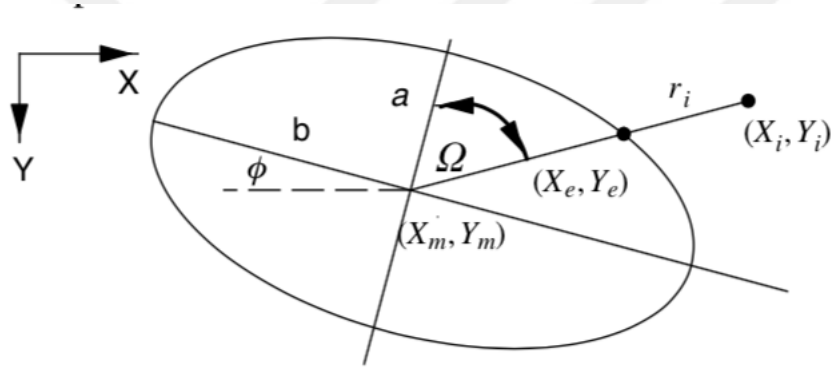
Mobil kamera üzerinden bardak tanımlama işleminin başarılı olarak gerçeklemesi sonrası bardak ağzı belirleme süreçleri başlatılır. Hedef bardağın tam üzerinden bakılması dışında bardak ağzı kontur olarak elipse benzer olacaktır. Bu, bir dairenin perspektif dönüşümü ile bir elips biçimine dönüşmesidir. Burada elde edilen geometrik şekil dönüşümü Şekil IV.35'te gösterildiği gibi olacaktır.



Şekil IV.35 Daire ve elips eksen dönüşümü

Gözlemlenen elipsin merkezinin ve eksenlerinin, orijinal dairenin merkezinin ve eksenlerinin perspektif haritalamasıyla elde edilen görüntüler olmadığı anlamına gelir.

Bir daireyi perspektif bir dönüşümle deforme etmek, bir elips verir. Ancak uygulamada eksenler ve merkez, orijinal dairenin eksenleri ve merkezi çakışmayabilir. Şekil IV. 36'da ve Tablo IV.1'de elips parametreleri gösterilmektedir.

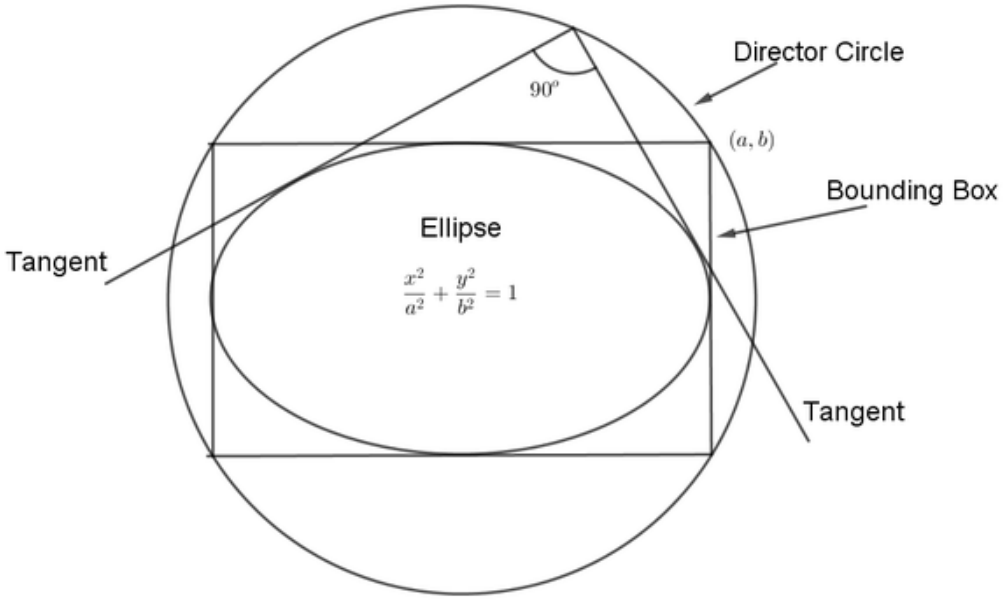


Şekil IV.36 Elipsin parametreleri gösterimi

Tablo IV.1 Elips Parametreleri

(Xe, Ye)	Elipse noktaları
(Xm, Ym)	Elipse merkezi
a, b	Yarım eksen uzunluğu
ϕ	Koordinat eksenlerine göre yarım eksen açısı

Bardak ağzında belirlenen elips görüntüsünün çiziminde sonraki paragrafta verilen elips işlemleri için mobil cihaz üzerinde bulunan ivme ölçer sensörü (Accelerometer) kullanılmıştır.



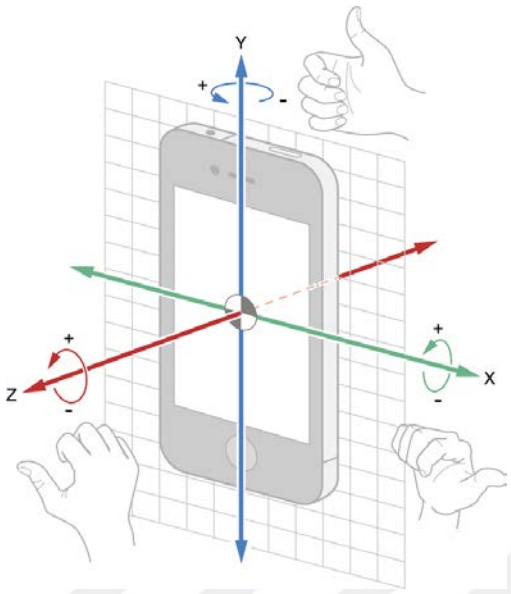
Şekil IV.37 Daire içindeki elipse çember denklemi

İvme Ölçerin mobil cihazın konumlarına karşın F değerleri ürettiği Tablo IV.2'de gösterilmektedir.

Tablo IV.2 İvme Ölçer Konumları

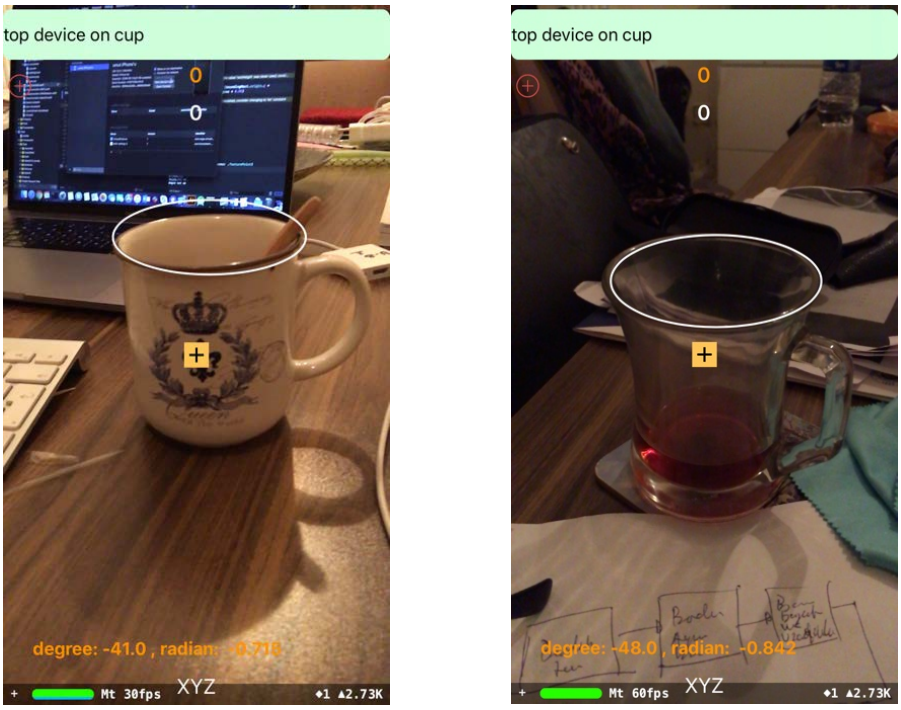
F = 0	Telefon masaya dik
F = -100	Telefon masanın üzerinde, kamera yukarıda
F = 100	Telefon masanın üzerinde, kamera masaya dayalı

Burada telefonun bazı konumları kullanışlı olmayacağından F'nin mutlak değeri kullanılmaktadır. Bu durumda F değeri 0-100 arası değerler alır. Daire de 2a=2b'dir. F'nin değeri a'nın bir katsayısı olarak değerlendirilmesi ile daireden elipse geçiş yapılır. Buna göre F=0 olduğunda a=0 olur ve daire çizgi görünümünde, F=100 olduğunda ise a=b olarak daire tam daire konumunda olacaktır.



Şekil IV.38 İvme Ölçer sensörü

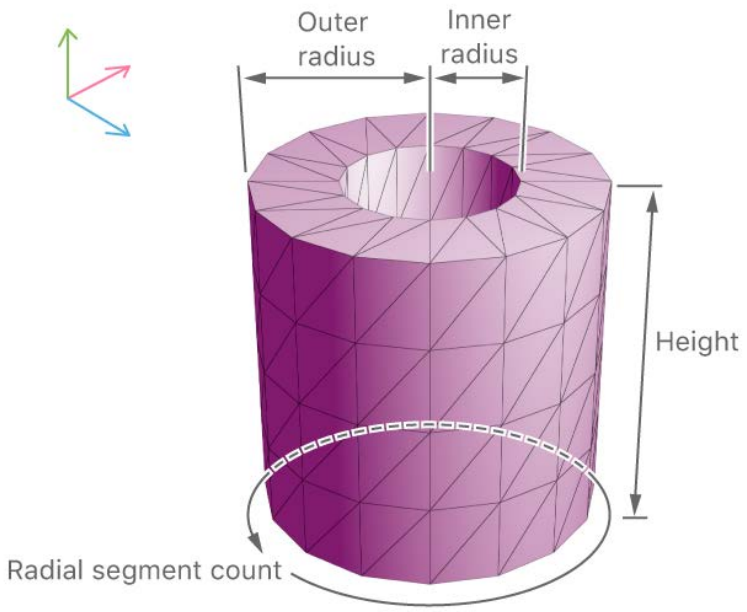
İvme Ölçer sensöründen alınan bilgilerle yukarıda açıklanan matematiksel işlemlere hesaplanan ve ekrana çizilen bardak ağzı çizimi Şekil IV. 39'da gösterilmektedir.



Şekil IV.39 Bardak ağzı çizimi ekran görüntüsü

4.2.3. Bardak Boyutu ve Konumu Gerçeklemesi

3 boyutlu uzayda tespit edilen bardak yerine konumlandırılacak olan geometrik tüp şekli için tespit edilen bardağın ağız genişliği ve yüksekliği bu aşamada hesaplanmaya çalışılmaktadır. Şekil IV.40'da SCNTube nesnesi gösterilmektedir.



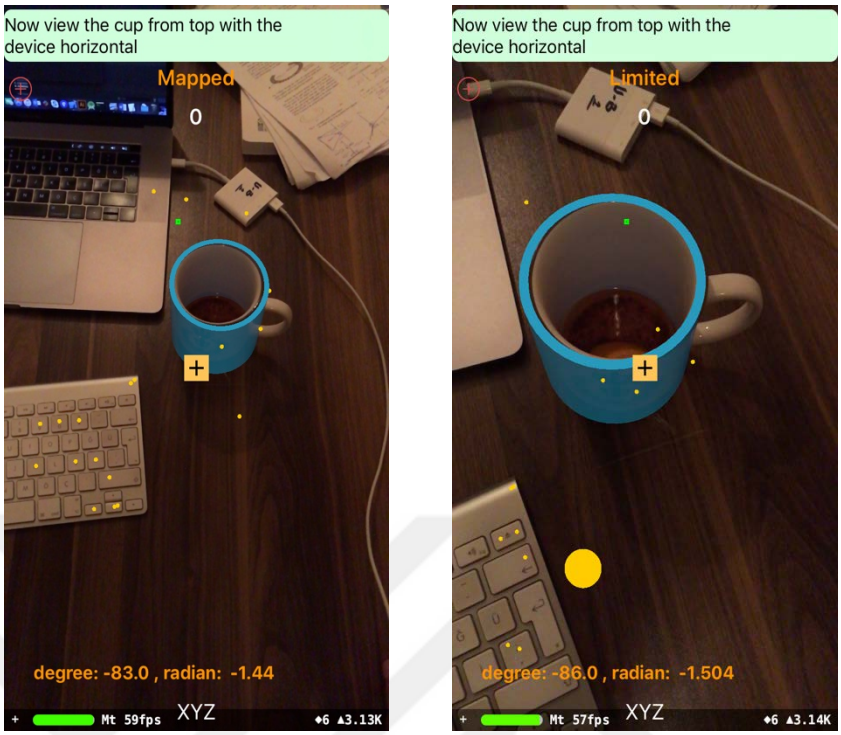
Şekil IV.40 SCNTube nesnesi

Öncelikle bardak ağzı boyutunun hesaplanması için kullanıcıdan tespit edilen bardağa tam yukardan, cihaz yatay konumda olarak bakılması istenmektedir. Şekil IV.41'de bu işlem gösterilmektedir. Bardağa yukarıdan bakma işlemi sırasında hesaplamanın doğru bir şekilde kontrollü olarak gerçekleştirilmesi için yine cihazın üzerinde bulunan ivme ölçer sensörü kullanılarak yaklaşık 88° - 90° arasında olduğu durumda hesaplama işlemi gerçekleşmektedir. Bu kontrol sağlandığı durumda OpenCV kütüphanesi

üzerindeki Hough Circle Transform fonksiyonu kullanılmıştır [44]. Burada iki boyutlu düzlemde bardak merkez noktası ve bardak çapı elde edilmektedir. Elde edilen daire merkezi üzerinden X,Y düzleminde X değerleri sabit tutularak Y1,Y2 değerleri elde edilerek oluşan X1,Y1 ve X2,Y2 değerleri 3 boyutlu koordinat düzleminde Z değerlerine mevcut Z değerleri sabit tutulmuştur ve böylece yaklaşık olarak bardak ağzı hesaplanmaktadır. Şekil IV.41'de bardak ağzı hesaplanmış ve merkez noktası bulunmuş işlem gösterilmektedir.

Şekil IV.41 OpenCV ile bardak merkezinin bulunması

Tespit edilen bardak merkezi ve bardak ağzı genişliğinde hesaplanan ölçeklerde Şekil IV.42'deki gibi 3 boyutlu bir tüp yerleştirilmiştir.



Şekil IV.42 Tüp yerleştirilmesi gösterimi

4.2.4. Oyun Oynama Evresinin Gerçeklemesi

Kamera ortamından algılanan örüntü düzleminde tespit edilen bardağın ölçeğinde 3 boyutlu bir tüp yerleştirilmesi işleminden sonra kullanıcının bardak ile fiziksel etkileşim evresi yani oyun evresi başlamış olur. Kullanıcı bulunduğu konum ve bakış açısına göre bardakla etkileşimini bardağa top atarak sağlamaktadır. Ekrana basılı tutma süresi kullanıcının topu ne kadar şiddetli bir şekilde fırlatacağını belirtir. Bunun için zamanlayıcı kullanılarak ekrana dokunma süresinden bir top atma şiddeti hesaplanır. Şekil IV.43'te örnek program kodları gösterilmiştir.

```
    override func touchesBegan(_ touches: Set<UITouch>, with event: UIEvent?) {
        if self.isAddedTube == true {
            timer.perform(closure: { () -> NextStep in
                self.power = self.power + 1
                return .continue
            })
        }
    }

    override func touchesEnded(_ touches: Set<UITouch>, with event: UIEvent?) {
        if self.isAddedTube == true {
            self.timer.stop()
            self.shootBall()
            print("power : \(self.power)")
        }
        self.power = 1
    }
```

Şekil IV.43 Top atış şiddeti hesaplaması

Oyunlar için gerçek görsellere yaklaşık bir geometri kullanmak oldukça iyi sonuç vermektedir. Fiziksel çarpışmaların doğruluk derecesine dair bir sınırlama vardır. Dolayısıyla, çarpışma oluşturma için kullanılan tam bir geometriyi kullanmak yerine, basitleştirilmiş bir yaklaşık şekil çarpışma tespiti için yararlı olur. Oyunlar için, nesne karmaşıklığından bağımsız olarak oyun nesnesini temsil eden küre ve kutu gibi basit geometrik sanal (nesneler) şekillere güvenmek yaygın bir çözüm tercihidir. Sanal nesneler çarpışırsa, gerçek nesnelerin de çarpıştığı varsayılır. Sınırlayıcı hacimler genellikle geometriyi tam olarak sarmak için yapılır. Statik görüntü oluşturma verileri malzemeye göre sıralanabilirken, çarpışma verileri genellikle mekansal olarak düzenlenir [45].

Fiziksel simülasyon için ARKit kütüphanesinde bünyesindeki fizik simülasyonları gerçekleştiren “SCNPhysics” sınıfları kullanılmıştır. Bu fizik simülasyonlarında Uluslararası Birimler Sistemi (System of Units) değerleri kullanılmaktadır. Kütle birimi kilogramdır; kuvvet, itme ve tork birimleri,

newton, newton saniye ve newtonmetredir. Düğüm pozisyonları ve boyutları için mesafe birimi ise metredir.

Çarpışma tespiti çoklu geometrik bir yapıya sahip bir alandan oluşmaktadır. Örneğin, bir dünya simülasyonunda hem dünyanın hem de o dünyanın nesneleri çokgenler, küreler ve kutular gibi geometrik varlıklar olarak temsil edilir.

Kurguladığımız oyunda bardak yerine yerleştirilecek hedef nesnemiz tüp, top için küre ve tüp ağzında (tüp çapında) ise başarılı bir atış için topun (küre) etkileşime geçeceği bir düzlem (plane) gibi geometrik şekiller kullanılmaktadır.

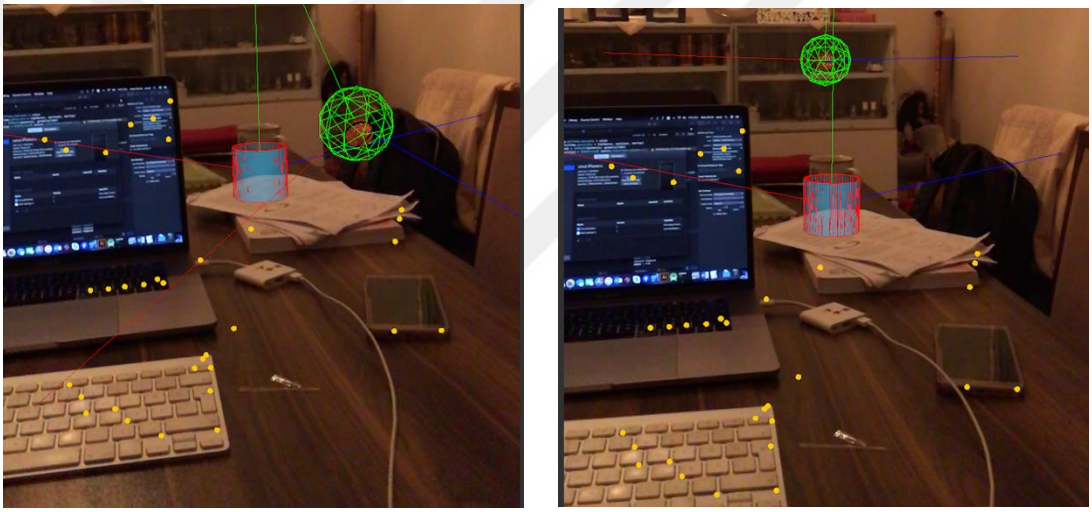
Kullanılan geometrik nesnelerin simülasyondaki kuvvetlerle ve diğer geometrik nesnelerle nasıl etkileşime girdiğini belirleyen tip (type) özelliği bulunur. Statik cisimler kuvvetlerden ve çarpışmalardan etkilenmez ve hareket edemezler. Dinamik cisimler diğer vücut tipleri ile kuvvetlerden ve çarpışmalardan etkilenir. Bu iki özelliğin dışında kinematik (kinematic) olarak adlandırılan bir özellik daha mevcuttur. Kinematik özelliği ile dinamik bir gövdeye kuvvet veya tork uygulama sonucunda, kütlesi ile orantılı bir hızlanma (veya açısal hızlanma) ile sonuçlanır.

Kullanılan tüp ve düzlem şekilleri statik, küre şekli ise fiziksel olarak dinamik olarak işaretlenmiştir.

Bir fizik gövdesi, kendi kategorilerini ilan etmenin yanı sıra, hangi fizik gövdesi kategorileri ile etkileşime girdiğini de ilan eder. Küre, düzlem ve tüp

şekli ile etkileşime geçer. Düzlem, küre ile tüp ise küre ile karşılıklı bir etkileşim içerisindedir.

Kullanıcı kamera bakış açısını ile yukarı aşağı hareket ettirerek top atma yönünü belirleyecektir. Böylelikle 3 boyutlu uzayda bardak konumu ile etkileşime geçtiği durumda, topun bardağa çarpmasıyla ya da bardağın içine girmesiyle etkileşim gerçekleşmiş olur. Şekil IV.44'te kullanıcının topu fırlatma örneklendirilmesi gösterilmektedir.



Şekil IV.44 Kullanıcının bardak ile etkileşimi

V. SONUÇLAR

Bu çalışmada; Derin Öğrenme ile Artırılmış Gerçeklik uygulamalarının tek bir üründe birleştirilmesi sağlanmıştır. Burada; YOLO gibi bir sinir ağı nesne tespiti, çevredeki alanı ve cihazın bulunduğu yeri algılamada kullanılmıştır.

İlk zorluk, büyük miktarda ilgili verinin toplanması ve iyi bir şekilde bir sinir ağını eğitmek üzere bu verilerin ön işlemlerden geçirilmesidir. Ancak tam ölçekli bir ürün için yine de çok fazla veri gerekir. Bazı işlemleri kolaylaştırabilecek pek çok kütüphane (AG, Derin öğrenme, nesne algılama vb.) mevcut olmasına rağmen gerçek zamanlı çalışan bir mobil oyun da kullanılmasında zamanlanma sorunları da vardır.

Özellikle örüntü tanıma ve örüntü işleme de daha başarılı sonuçlar veren algoritmalar oldu. Fakat bu algoritmaların işletilmesinde mobil işlemcinin bir bilgisayara göre kısıtlı işlem gücüne sahip olmasından dolayı üretilecek görüntülerde gecikmeler olacaktır. Bardağın ağzına çizilen elips için bu tez kapsamında farklı algoritmalar denenmiştir. Örneğin; RANSAC algoritması elipsin tanınması konusunda bunlardan biriydi. Fakat başarılı sonuç alabilmesi için iterasyon sayısının fazla olması ve işlem sonuçlanana kadar 1-2sn. gibi süreler geçmesi, bu algoritmayı kullanışsız hale getirmiştir [46]. Bu nedenle neredeyse tüm işlemler için yeniden değerlendirmek ve programlamak zorunlu olmuştur. Sonuç olarak, bardağın ağzına bir elips çizilmesi için ivme ölçeri kullanarak hızlı sonuç veren bir yöntem geliştirilmiştir.

Aynı şekilde sahnede olan 3D nesnelerin daha gerçekçi ölçeklendirilmesi ve modellenmesi için Structure From Motion algoritması tercih edilebilirdi [47].

ARKit 2.0 piyasaya sürüldüğünden beri, bir AR deneyimini başka bir kullanıcıyla paylaşmak mümkündür. Bu çalışmada birden fazla oyuncuya aynı deneyimi paylaştırmak ek bir özellik olarak kullanılabilir. Burada birden fazla oyuncunun eş zamanlı olarak aynı sahne üzerinde oyunu oynamasını mümkün kılar. İki oyuncudan biri bardak tespiti ve tanıma işlemini gerçekleştirir daha sonrasında diğer oyuncuyu davet ederek bu tespit edilen verilerin ikinci oyuncunun kamerasına aktarılması ile eş zamanlı bir yarış ortamı sağlanabilir. Bu ortam içinde kullanıcılara atış mesafesinin uzaklığı ve yakınlığı, atış başarısı, rakip oyuncunun topu ile etkileşime geçerek durdurma gibi birçok ek özellik ile oyun daha da zenginleştirilir.

Tek kullanıcılı aynı oyunda deneyim bir çöp kovası ya da benzer nesneler ile de gerçekleştirilebilinir. Bunun için çeşitli çöp kovası görselleri ile bir model eğitilir ve uygulamaya eklenir. Hedef bardak ile oyuncu arasında engel nesneleri konulabilir ve bununla ilgili zorluk seviyeleri belirlenerek özellikler listesine dahil edilebilir.

VI. KAYNAKLAR

[1] K. T. M. B. M. G. H. Kato, «A registration method based on texture tracking using ARToolKit,» %1 içinde *2003 IEEE International Augmented Reality Toolkit Workshop*, Tokyo, Japan, 2003.

[2] Wikitude GmbH, «Wikitude,» https://www.wikitude.com/, 2019.

[3] P. Mistry, «The Thrilling Potential Of Sixth Sense Technology,» %1 içinde *TEDIndia*, India, 2009.

[4] D. D. İ. Y. O. Güler, «Artırılmış Gerçeklik: Montaj ve Bakım Uygulamalarında El Tanıma Teknolojisi ile Etkileşim Çalışmaları,» %1 içinde *ab2018*, Karabük, 2018.

[5] Y. B. a. G. H. Y. LeCun, «Deep learning,» *Nature*, cilt vol. 521, p. pp. 436–444, 2015.

[6] B. D. H. H. B. A. Şeker, «Derin Öğrenme Yöntemleri ve Uygulamaları Hakkında Bir İnceleme,» *Gazi Mühendislik Bilimleri Dergisi*, cilt 3, p. pp. 47–64, 2017.

[7] D. H. H. a. T. N. Wiesel, «Receptive fields and functional architecture of monkey striate cortex,» *J. Physiol.*, cilt vol. 195, no. no. 1, p. pp. 215–243, 1968.

[8] Y. B. I. G. A. C. M. Nielsen, «Deep Learning Book,» http://neuralnetworksanddeeplearning.com/, 2017.

[9] E. Ü. Ö. İnik, «Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri,» *Gaziosmanpasa Journal of Scientific Research,* cilt 6, no. 3, pp. 85-04, 2017.

[10] MathWorks, «Convolutional Neural Network 3 things you need to know,» https://www.mathworks.com/solutions/deep-learning/convolutional-neural-network.html.

[11] D. A. A. S. Dr. K. Kayaalp, «Derin öğrenme ve Türkiye'deki uygulamaları,» *IKSAD International Publishing House,* Cilt %1 / %2ISBN vol. 978-605-7510-53-2, p. 12–24, Sep. 2018.

[12] LeNet-5, «LeNet-5, convolutional neural networks,» http://yann.lecun.com/exdb/lenet/, 1999.

[13] AlexNet, «ImageNet Classification with Deep Convolutional Neural Networks,» NIPS Proceedings, http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks, 2012.

[14] ZFNet, «Visualizing and Understanding Convolutional Networks,» Matthew D Zeiler, Rob Fergus, https://arxiv.org/abs/1311.2901 , 2013.

[15] GoogLeNet/Inception, «Going Deeper with Convolutions,» Google, 2015. [Çevrimiçi]. Available: https://ai.google/research/pubs/pub43022.

[16] VGGNet, «Very Deep Convolutional Networks for Large-Scale Image Recognition,» K. Simonyan, A. Zisserman, https://arxiv.org/abs/1409.1556 , 2014.

[17] ResNet, «Deep Residual Learning for Image Recognition,» https://arxiv.org/abs/1512.03385, 2015.

[18] DarkNet, «Darknet: Open Source Neural Networks in C - Joseph Redmon,» pjreddie, https://pjreddie.com/darknet/, 2018.

[19] Kaggle, «The State of Data Science & Machine,» Kaggle, https://www.kaggle.com/surveys/2017, 2017.

[20] A. A. P. B. E. B. Z. C. C. C. G. S. C. A. D. J. D. M. D. S. G. I. G. A. H. G. I. M. I. Y. J. R. J. L. K. M. K. J. L. D. M. R. M. S. M. D. M. Abadi, «TensorFlow: Large-Scale Machine Learning on Heterogeneous Distributed Systems,» Google, 01 05 2017. [Çevrimiçi]. Available: https://www.tensorflow.org/.

[21] E. S. J. D. S. K. J. L. R. G. S. G. T. D. Y. Jia, «Caffe: Convolutional Architecture for Fast Feature Embedding,» *Proceeding MM '14 Proceedings of the 22nd ACM international conference on Multimedia*, pp. 675-678, 2014.

[22] Caffe2, «Facebook Open Source, A New Lightweight, Modular, and Scalable Deep Learning Framework,» Caffe2, https://caffe2.ai.

[23] C. F. K. K. R. Collobert, «NIPS Workshop on Machine Learning Open Source Software,» Torch | Scientific computing for LuaJIT, 01 05 2017. [Çevrimiçi]. Available: http://torch.ch/.

[24] Keras, «Keras Deep Learning Library,» Chollet, F, https://keras.io/, 10.10.2017.

[25] M. L. Y. L. M. L. N. W. M. W. T. X. B. X. C. Z. Z. Z. T. Chen, «MXNet: A Flexible and Efficient Machine Learning Library for Heterogeneous Distributed Systems,» %1 içinde *Workshop on Machine Learning Systems In Neural Information Processing Systems*, 2016.

[26] A. A. F. Seide, «CNTK: Microsoft's Open- Source Deep-Learning Toolkit,» %1 içinde *Proceeding KDD'16 Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, 2135-2135*, 2016.

[27] Deeplearning4j, «Distributed Deep Learning for the JVM,» Skymind, https://deeplearning4j.org/, 10.10.2017.

[28] D. Yuret, «Welcome to Knet.jl's documentation!,» 06 08 2016. [Çevrimiçi]. Available: http://denizyuret.github.io/Knet.jl/latest/.

[29] Theano Development Team, «Theano: A {Python} framework for fast computation of mathematical expressions,» *ArXiv e-prints*, cilt abs/1605.02688, 2016.

[30] TuriCreate Library, «Turi Create simplifies the development of custom machine learning models,» https://github.com/apple/turicreate, 2016.

[31] M. Z. M. a. B. A. Graham, «Augmented reality in urban places: contested content and the duplicity of code,» 10 August 2012. [Çevrimiçi]. Available: https://rgs-ibg.onlinelibrary.wiley.com/doi/abs/10.1111/j.1475-5661.2012.00539.x.

[32] T. R. Gayathri, R. P. Aneesh ve G. R. Nayar, «Feature based simultaneous localisation and mapping,» %1 içinde *2017 IEEE International Conference on Circuits and Systems (ICCS)*, Thiruvananthapuram, India, 20-21 Dec. 2017.

[33] ARKit, «ARKit Framework Overview,» Apple, https://developer.apple.com/documentation/arkit, 2019.

[34] ARCore, «ARCore Overview,» Google, https://developers.google.com/ar/, 2018.

[35] Vuforia, «Vuforia Augmented Reality,» PTC, https://www.ptc.com/en/products/augmented-reality, 2019.

[36] DeepAR, «DeepAR Augmented Reality SDK,» I Love IceCream Ltd., https://www.deepar.ai/augmented-reality-sdk, 2019.

[37] TuriCreate–MxNet, «Advanced Deep Learrning with MxNet,» https://turi.com/learn/userguide/supervised-learning/advanced_deeplearning_mxnet.html, 2017.

[38] Vision, «Vision Framework Overview,» Apple, 2019. [Çevrimiçi]. Available: https://developer.apple.com/documentation/vision .

[39] CoreML, «CoreML Framework Overview,» Apple, https://developer.apple.com/documentation/coreml, 2019.

[40] OpenCV IOS, «https://docs.opencv.org/2.4.13.7/doc/tutorials/ios/table_of_content_ios/table_of_content_ios.html,» OpenCV.

[41] IMAGENET, «Image Database,» Stanford Vision Lab, Stanford University, Princeton University, http://image-net.org/2016, 2016.

[42] VIA, «VGG Image Annator,» Visual Geometry Group, Oxford University, http://www.robots.ox.ac.uk/~vgg/software/via/, 2017.

[43] TuriCreate, «Object Detection, Data Acquistion,» https://apple.github.io/turicreate/docs/userguide/object_detection/, 2017.

[44] OpenCV, «Hough Circle Transform Theory – Image Processing in OpenCV,» https://docs.opencv.org/3.4.2/da/d53/tutorial_py_houghcircles.html.

[45] C. Ericson, Real-Time Collision Detection, Series in interactivite 3D technology, Morgan Kaufmann Publishers is an imprint of Elsevier, 2005.

[46] RANSAC, «Using RANSAC for estimating geometric transforms in computer vision,» https://www.mathworks.com/discovery/ransac.html, 2019.

[47] Structure From Motion, «Structure from motion (SfM) is the process of estimating the 3-D structure of a scene from a set of 2-D images,» https://www.mathworks.com/help/vision/ug/structure-from-motion.html., 2019.