DERİN ÖĞRENME TABANLI ŞEFFAF NESNE TANIMA

Korhan MUTLUDOĞAN

T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME TABANLI ŞEFFAF NESNE TANIMA

Korhan MUTLUDOĞAN
0000-0002-2387-0944

Dr. Öğr. Üyesi Metin BİLGİN
(Danışman)

YÜKSEK LİSANS TEZİ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

BURSA – 2020

TEZ ONAYI

Korhan MUTLUDOĞAN tarafından hazırlanan "DERİN ÖĞRENME TABANLI ŞEFFAF NESNE TANIMA" adlı tez çalışması aşağıdaki jüri tarafından oy birliği ile Bursa Uludağ Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği Anabilim Dalı'nda **YÜKSEK LİSANS TEZİ** olarak kabul edilmiştir.

Danışman :Dr. Öğr. Üyesi Metin BİLGİN

Başkan	:	Dr. Öğr. Üyesi Metin BİLGİN 0000-0002-4216-0542 Bursa Uludağ Üniversitesi, Mühendislik Fakültesi, Bilgisayar Yazılımı Anabilim Dalı	İmza
Üye	:	Dr. Öğr. Üyesi Cengiz TOĞAY 0000-0001-5739-1784 Bursa Uludağ Üniversitesi, Mühendislik Fakültesi, Siber Güvenlik Anabilim Dalı	İmza
Üye	:	Dr. Öğr. Üyesi Kazım YILDIZ 0000-0001-6999-1410 Marmara Üniversitesi, Teknoloji Fakültesi, Bilgisayar Bilimleri Anabilim Dalı	İmza

Yukarıdaki sonucu onaylarım

Prof. Dr. Hüseyin Aksel EREN
Enstitü Müdürü
10/07/2020

Fen Bilimleri Enstitüsü, tez yazım kurallarına uygun olarak hazırladığım bu tez çalışmasında;

- tez içindeki bütün bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- görsel, işitsel ve yazılı tüm bilgi ve sonuçları bilimsel ahlak kurallarına uygun olarak sunduğumu,
- başkalarının eserlerinden yararlanılması durumunda ilgili eserlere bilimsel normlara uygun olarak atıfta bulunduğumu,
- atıfta bulunduğum eserlerin tümünü kaynak olarak gösterdiğimi,
- kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- ve bu tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı

beyan ederim.

10/08/**2020**

Korhan MUTLUDOĞAN

ÖZET

Yüksek Lisans Tezi

DERİN ÖĞRENME TABANLI ŞEFFAF NESNE TANIMA

Korhan MUTLUDOĞAN

Bursa Uludağ Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

Danışman: Dr. Öğr. Üyesi Metin BİLGİN

Gelişen teknoloji ve yapay zekâ algoritmalarıyla birlikte nesne tanıma uygulamaları hayatımızın içinde yerini almaya başlamıştır. Günlük hayatımızda plaka tanıma, optik karakter tanıma gibi uygulamalar artık hayatımızın vazgeçilmezi haline gelmiştir. Günümüzde devam eden teknolojik gelişmelere paralel olarak güvenlik kameralarından şüpheli durum tespiti veya otonom araçlar gibi yakın gelecekte hayatımızla iç içe olacak teknolojilerin gelişimi hızla artmaktadır. Nesne tanıma alanında gerçekleştirilen çalışmalar genel olarak şeffaf olmayan nesneler üzerinedir ve şeffaf nesnelerle ilgili çalışma sayısı çok sınırlı kalmaktadır. Oysa çevremizde şeffaf olmayan nesneler kadar şeffaf nesneler de bulunmaktadır ve şeffaf nesnelerin tanınması da önem taşımaktadır. Şeffaf nesne tanımanın, yakında hayatımızın içinde yer alacak robotların çevresini algılamasına ve geri dönüşüm tesislerindeki nesnelerin ayıklanma sürecine katkısı olacaktır.

Bu çalışmada, şeffaf ve şeffaf olmayan bardaklardan oluşan bir veri kümesi ile eğitilen bir sistem vasıtasıyla bardakların şeffaflığı tespit edilmeye çalışılmıştır. Önerilen sistemin geliştirilmesinde son zamanlarda tanıtılan yeni bir derin öğrenme yaklaşımı olan kapsül ağları kullanılmıştır. Elde edilen sonuçların karşılaştırılabilmesi için, aynı veri kümesi LeNet, AlexNet ve ResNet modelleri üzerinde çalıştırılmıştır. Çalışmanın sonuçları değerlendirildiğinde kapsül ağlarının, sınıflandırma doğruluğunda, kullanılan diğer derin öğrenme yöntemlerinden daha iyi sonuçlar elde ettiği görülmüştür. Sınıflandırma doğruluğuna göre diğer yöntemler AlexNet, ResNet ve LeNet şeklinde sıralanmıştır. Çalışma sonucunda kapsül ağlarının şeffaf nesne tanıma probleminde kullanılabileceği ve mevcut yöntemlerden daha yüksek doğruluk oranlarına ulaştığı görülmüştür.

Anahtar Kelimeler: Bilgisayarla görme, nesne tanıma, şeffaf nesne tanıma, derin öğrenme, danışmanlı öğrenme, kapsül ağları

2020, vii + 69 sayfa.

ABSTRACT

MSc Thesis

DEEP LEARNING BASED TRANSPARENT OBJECT DETECTION

Korhan MUTLUDOĞAN

Bursa Uludağ University
Graduate School of Natural and Applied Sciences
Department of Computer Engineering

Supervisor: Dr. Metin BİLGİN

Object detection applications started to take their places in our lives with the improvements in technology and artificial intelligence. In daily life, some applications such as license plate detection, optical character recognition become indispensable. In parallel with ongoing technological developments today, the technologies which soon will be a part of our daily life such as detection of suspicious situations through security cameras and autonomous cars, are improving rapidly. The studies in the object detection area are have generally focused on opaque objects. The number of studies on transparent objects is very limited. However, there are transparent objects as well as opaque objects around us, and the detection of transparent objects is also important. Transparent object detection will contribute to the perception of the environment of the robots that will soon take place in our lives and the sorting process of the objects in the recycling facilities.

In this study, a system which is trained by the dataset that contains transparent and non-transparent glasses is used to detect transparency of these glasses. Recently introduced deep learning approach capsule networks are used to develop a proposed system. To compare the obtained results, LeNet, AlexNet, and ResNet are trained and tested with the same dataset. When the result of the study is evaluated, it was seen that CapsNet got better results on classification accuracy than other deep learning methods that are used in this study. According to the classification accuracy, other methods were lined up as AlexNet, ResNet, and LeNet. As a result of the study, it has been seen that capsule networks can be used in transparent object detection problems and reach higher accuracy rates than existing methods.

Keywords: Computer vision, object detection, transparent object detection, deep learning, supervised learning, capsule networks

2020, vii + 69 pages.

ÖNSÖZ VE TEŞEKKÜR

Günümüzde yeni bir sanayi devrimi yapay zekâ alanında yaşanmaktadır. Önceki devrimlerde bunu yakalayamayan toplumlar geri kalırken bu devrimi kaçıran toplumlar belki de yok olma tehlikesiyle karşı karşıya kalacaktır. Bu sebeple, dev fabrikalar ve kompleks makineler yerine bir bilgisayar ve bir beyne ihtiyaç duyan bu devrimi yakalamak ülkemiz için de çok önemlidir. Bu çalışma, yapay zekâ ve bilgisayarla görme alanındaki gelişmelere katkı sağlamak amacıyla yapılmıştır.

Bu tez çalışmasını yaparken saat gözetmeksizin her türlü soruma cevap veren, tıkandığım noktalarda yardımcı olan, gerektiği yerde motive eden danışmanım Dr. Öğr. Üyesi Metin BİLGİN'e teşekkürlerimi sunarım.

Bu süreç boyunca uygun çalışma ortamını oluşturmam için desteğini esirgemeyen aileme ve tüm sıkıntılara hoşgörü gösteren ve tüm zamansızlığıma sabreden müstakbel eşim Sezin KOKUM'a çok teşekkür ederim.

Son olarak, bilim ve teknolojide bugünlere gelmemize katkısı olan, doğruları söylemek pahasına gerektiğinde toplumlarının dogmalarıyla ters düşen, dışlanan, işkence gören, hatta öldürülen tüm bilim insanlarına teşekkürler.

Korhan MUTLUDOĞAN
10.08.2020

İÇİNDEKİLER

Sayfa

SİMGELER ve KISALTMALAR DİZİNİ

Simgeler	Açıklama
A	Doğruluk (Accuracy)
a	Aksiyon (Action)
B	Davranış (Behaviour)
i	Girdi (Input)
P	Kesinlik (Precision)
R	Duyarlılık (Recall)
r	Pekiştirme Sinyali (Reinforcement Signal)
s	Durum (State)

Kısaltmalar	Açıklama
CapsNet	Kapsül Ağı (Capsule Network)
DN	Doğru Negatif
DP	Doğru Pozitif
FS	F-Skoru (F-Score)
HOG	Yönlü Gradyenlerin Histogramı (Histogram of Oriented Gradients)
ILSVRC	ImageNet Büyük Ölçekli Görsel Tanıma Yarışması (ImageNet Large Scale Visual Recognition Challenge)
ReLU	Doğrultulmuş Lineer Ünite (Rectified Linear Unit)
ResNet	Artık Ağ (Residual Network)
RGB	Kırmızı Yeşil Mavi (Red Green Blue)
SIFT	Ölçek Değişimsiz Özellik Dönüşümü (Scale Invariant Feature Transformation)
SVC	Destek Vektör Sınıflandırıcı (Support Vector Classifier)
SVM	Destek Vektör Makinesi (Support Vector Machine)
TB	Tam Bağlantılı (Fully Connected)
TN	Toplam Negatif
TP	Toplam Pozitif
YN	Yanlış Negatif
YP	Yanlış Pozitif

ŞEKİLLER DİZİNİ

Sayfa

ÇİZELGELER DİZİNİ

Sayfa

1. GİRİŞ

Nesne tanıma problemi, yapay zekâ ve bilgisayarla görme alanlarında uzun zamandır üzerine birçok çalışma yapılan bir problemdir. Bu problemin çözümü için kullanılan yöntemlerin bazıları arka plan çıkarma (Gupte ve ark. 2002) ve Yönlü Gradyenlerin Histogramı (Histogram of Oriented Gradients, HOG) ile nesne tanıma (Dalal ve Triggs 2005) gibi yöntemlerdir. El yazısı rakamları tanımak için LeNet (LeCun ve ark. 1998) yöntemi geliştirilmiş, AlexNet (Krizhevsky ve ark. 2012) ve Artık Ağ (Residual Network, ResNet) (He ve ark. 2016) gibi modellerin gelişmesiyle birlikte nesne tanımada derin öğrenme yöntemleri kullanılmaya başlanmıştır. Kapsül Ağları (Capsule Networks, CapsNet) (Sabour ve ark. 2017), nesne tanıma problemine yeni ve etkili bir çözüm getirmiştir.

Nesne tanıma problemi için pek çok farklı yöntem denenmiş ve başarılı sonuçlar alınmış olmasına rağmen bu çalışmaların çoğu şeffaf olmayan nesneler üzerine yoğunlaşmıştır. Şeffaf olmayan nesnelere kıyasla şeffaf nesneler için çok az sayıda çalışma gerçekleştirilmiştir. Şeffaf nesnelerin makineler tarafından algılanabilmesi günlük hayatta karşılaşılabilecek pek çok soruna yeni çözümler sunulmasını sağlayabilir. Örneğin; geri dönüşüm tesislerinde nesnelerin ayıklanmasını ve günlük hayatımıza yeni yeni girmeye başlayan otonom robotların çevredeki nesneleri algılamasını kolaylaştırmak adına şeffaf nesne tanımaya bulunacak çözümler büyük önem arz etmektedir. Şeffaf nesne tanıma probleminin çözümünde daha önce, kolektif ödül tabanlı yaklaşım (Kompella ve Sturm 2011), kızılötesi ışık kullanarak tespit etme (Klank ve ark. 2011) gibi çeşitli yöntemler kullanılmıştır. Birtakım derin öğrenme yöntemleri ile de bu problem çözülmeye çalışılmıştır (Khaing ve Masayuki 2018).

Bu çalışmanın amacı, su bardağı, kahve kupası gibi nesnelerin varlığını ve şeffaflığını bir kamera ile elde edilen görüntüler üzerinden tespit etmektir. Bu işlemi yaparken CapsNet gibi yeni derin öğrenme yöntemleri kullanılarak doğruluğun arttırılması amaçlanmış ve mevcut derin öğrenme yaklaşımlarıyla karşılaştırması yapılarak elde edilen sonuçlar sunulmuştur. Bu çalışma, halihazırda hayatımıza girmeye başlamış robot süpürgeler ve onlar gibi yakın gelecekte günlük hayatımızın bir parçası olması beklenen mobil

robotların çevrelerini algılamasını iyileştirmek için bir yöntem olarak kullanılabilir. Ayrıca geri dönüşüm tesislerindeki atıkların algılanması ve ayıklanması gibi potansiyel kullanım alanlarına da sahiptir.

2. KURAMSAL TEMELLER ve KAYNAK ARAŞTIRMASI

Zekâ; "İnsanın düşünme, akıl yürütme, objektif gerçekleri algılama, yargılama ve sonuç çıkarma yeteneklerinin tamamı, anlak, dirayet, zeyreklik, feraset" şeklinde tanımlanmıştır (Anonim 2020a). Fakat zekânın ne olduğu, binlerce yıldır çok farklı tanımlar ve metaforlar ile açıklanmaya çalışılmıştır. Tarihin ilk yazılı belgelerinden olan Sümer mitolojisinde, zekâ tanrısı Enki bulunmaktadır (Şekil 2.1). Dünyanın düzenlenmesi ile ilgili sekiz sütun, 466 satırlık bir tablette Enki kendinden "ülkelerin aklı ve kulağı" olarak bahsetmektedir (Çığ 2018).

Şekil 2.1. Sümer zekâ tanrısı Enki (Anonim 2016)

Cianciolo ve Sternberg'e (2004) göre zekânın doğası hakkında akıl yürüten ilk kişiler, psikologlar veya eğitimciler değil filozoflar olmuştur. Günümüzdeki üniversitelerin atası olan Akademi'nin kurucusu Platon'a göre aklı dayalı idealar dünyası ve duyularla algılanabilen fenomenler dünyası bulunmaktadır. Fenomenler dünyasındaki nesneler idealar dünyasındakilerin bir yansımasıdır (Akgündüz 2019). Gerçek bilgiler, gözlemlenebilir evrenin öncesinde, idealar evreninde bulunmaktadır (Birand 1958). Platon zekâyı, öğrenme kabiliyeti ile öğrenmeye ve bilgiye duyulan istek olarak tanımlamaktadır (Sternberg 1990). Platon'un öğrencisi olan Aristoteles, hocasının aksine

insanın zihninde doğuştan hiçbir ilke olmadığını savunur (Akbay 2017). Aristoteles zekâyı kıvrak zekâ ve çıkarım yeteneği olarak görmektedir (Sternberg 1990).

İlk İslam filozofu Kindi, aklı, hakikatlerini kavrayan basit bir cevher olarak tanımlamıştır (Uysal 2004). Aristoteles'ten etkilenmiş olan Kindi'nin de zekâyı, onun gibi çıkarım yapma yeteneği olarak gördüğü anlaşılmaktadır. Farabi'ye göre bilgi edinmenin iki yolu vardır. Bunlar; duygusal algı ve akıl yetisi ile bilgi edinmedir. Aklı tanımlarken de teorik ve pratik akıl olmak üzere iki sınıfa ayırmaktadır. Teorik akıl, doğuştan gelen bilgilerle edinilmiş akıl türüdür. Pratik akıl ise insanın tercihte bulunması veya kaçınması gereken durumları fark etmesini sağlayan akıldır (Kelleci 2011). İbn-i Sina ise zekâyı maddeden tümüyle soyutlanmış akli kavramları oluşturma yetisi olarak tanımlamış ve onu insanı diğer canlılardan ayıran en önemli özellik olarak görmüştür (Gürel ve Tat 2010).

Modern felsefenin babası kabul edilen Rene Descartes'a kadar akıl ve zekâ genellikle birbirine denk kavramlar olarak kullanılmıştır. Descartes'ın görüşleri ile bu kavramlar birbirinden ayrılmış, akıl; insani nitelikleri olduğu gibi ortaya koyabilen bir kavram olarak görülmüştür. Zekâ ise belli alandaki kabiliyetleri etkili bir şekilde ortaya koyabilme becerisi anlamında kullanılmaya başlanmıştır (Bayık 2019). John Locke'a göre zihin başlangıçta boş durumdadır ve bir şeyler yapabilmesi için duyusal girdilerin oluşması gerekmektedir. Aksi takdirde boş ve gelişmemiş olarak kalacaktır (Kılıç 2014). Akıl insana doğa tarafından verilmiştir ve ahlaki durumları da oluşturan doğa yasaları insanın aklı tarafından keşfedilir (Kılıç 2015). David Hume, Locke'un boş levha düşüncesine katılmaktadır. Hume'a göre akıl, kavrama ve bilgi yetisidir fakat bize sadece yol gösterir. Eyleme geçerken yapılan son tercih üzerinde tutkular etkili olur (Kırlı 2013). Kant, insanın kendi aklını başkalarının rehberliği olmadan kullanabilmesini aydınlanma olarak görmüştür. Ona göre doğru bilgi sadece deneysel tecrübelerle elde edilemez, önceden gelen bilgilerle birleştirilmesi gerekmektedir (Öktem 2004).

2.1. Yapay Zekâ

Antik çağlardan beri süregelen akıl ve zekâ üzerine fikir yürütme eylemleri sadece insanla sınırlı kalmamıştır. Akıllı makinelerle ilgili yapılan kurgular ve ortaya atılan fikirler de

neredeyse bir o kadar eskidir. İnsanlar eski zamanlardan beri, hayal güçlerinde kendileri gibi düşünen, konuşan, hareket eden canlı ve cansız yaratıklar oluşturmuşlardır. Bu sebeple mitolojilerde birçok insan zekâsına sahip cansız varlıkla karşılaşmak mümkündür. Bunlardan bazıları günümüzdeki gelişmelere bile ilham vermektedir. Homeros'un anlatılarına göre demircilikle uğraşan ve ateş tanrısı olan Hephaistos, kötürüm olmasından dolayı kendisine yardım etmeleri için altından, genç kadın gibi görünen zeki yardımcılar yaratmıştır (McCorduck ve ark. 1977). Yine Hephaistos tarafından Zeus'un emriyle, Girit'teki Europa'yı korumak için bronzdan bir dev olan Talos (Şekil 2.2) yaratılmıştır (Roberts 2018). Hephaistos'un yarattığı akıllı cansız varlıklar bununla sınırlı kalmamıştır. Ateşi çalıp yarattığı insanlara veren Prometheus'a sinirlenen Zeus, Hephaistos'tan insanları cezalandırmak için genç kadın görünümünde bir tuzak üretmesini ister. Böylece ortaya insanların arasına karışıp onlara sonsuz lanet getirecek olan Pandora ortaya çıkar (Mayor 2018). Sadece Yunan mitolojisinde değil Yahudi anlatılarında da kilden veya topraktan bir otomata olan Golem bulunmaktadır (Buchanan 2005).

Şekil 2.2. Eski bir Girit parası üzerinde Talos (Anonim 2010)

Tarihteki yapay zekâ örnekleri sadece mitlerde, efsanelerde, anlatılarda değil gerçek hayatta üretilen bazı mekanizmalarda da karşımıza çıkmıştır. Günümüzde bilgisayar

bilimlerindeki anlamıyla yapay zekâ olmasa bile mekanik olarak geliştirilen birtakım otomatalar mevcuttur. Çin'de Yan Shi isimli bir askeri makine ustası gerçek insan boyutunda ve görünümünde bir makineyi dönemin Chou kralı Mu'ya sunmuştur. Bu makinenin kendi kendine yürüme, şarkı söyleme, poz verme gibi birçok özelliği bulunmaktadır (Rosenberg 1992). İskenderiyeli Heron; buhar motorunun ilkel hali, basınçla otomatik açılan kapılar, rüzgâr enerjisiyle kendi kendine çalan org gibi mekanikler geliştirmiştir (Anonim 2020b). Ayrıca üzerinde çok ayrıntılı şekilde düşünülmüş, otomatik olarak hareket eden figürler ve platformlardan, patlayan alevlerden oluşan minyatür tiyatro sahneleri hazırlamıştır (McCorduck 2004). Leonardo da Vinci'nin ilham kaynağı olduğu düşünülen El-Cezeri, içecek servisi yapan otomatik garson, sifonlu el yıkama otomatı, suyun akışını kullanarak müzik çalan robot müzik grubu (Şekil 2.3) gibi otomatalar geliştirmiştir (Anonim 2020c). Saat yapımcısı Pierre Jaquet-Droz, 40 karaktere kadar olan mesajları kalem ve mürekkeple kâğıda yazabilen, bazı resimler çizebilen, 24 farklı nota kullanarak 5 farklı melodiden birini çalabilen insan görünümlü otomatlar üretmiştir (Deshpande 2015). Wolfgang von Kempelen'in "The Turk" isimli Türk görünümlü insansı bir mekanikten oluşan satranç oynayan mekanizmasının nasıl çalıştığı hakkında yoğun bir şekilde teoriler üretilmiş, daha sonra bunun mekanizmanın içinde saklanan bir satranç oyuncusunun verdiği komutlara göre hareket eden bir otomata olduğu ortaya çıkmıştır (McCorduck 2004).

Şekil 2.3. El-Cezeri'nin müzik otomatası (Anonim 2006)

Yapay zekâ konusunda yapılan ilk çalışma, McCulloch ve Pitts (1943) tarafından yapılan yapay bir nöronun tanıtıldığı çalışma olarak kabul edilmektedir. Bu çalışmada sinirsel aktivitedeki "ya hep ya hiç" karakteri ile bilgisayarlardaki 0 ve 1'ler arasında bir bağlantı kurulmuştur. Hesaplanabilir herhangi bir fonksiyonun bu nöronların ve mantıksal bağlantıların oluşturacağı bir ağ ile hesaplanabileceğini göstermişlerdir. Uygun şekilde oluşturulmuş ağların öğrenebileceğini söylemişlerdir. Hebb (1949) tarafından tanıtılan ve bugün "Hebbian öğrenme" olarak bilinen kural, nöronların arasındaki bağlantı güçlerinin değiştirmek için yapılacak basit güncellemeleri önermiştir.

Alan Turing (1950) yaptığı çalışmada "Makineler düşünebilir mi?" sorusunu sormuştur. Bu soru birçok kişi tarafından bilgisayar bilimlerindeki anlamıyla yapay zekânın başlangıcı olarak kabul edilmektedir. Aynı çalışmada bu sorunun cevabını anlamanın yolunu da Turing testi olarak bilinen yöntemle vermiştir. Turing testini oluştururken, imitasyon oyunu denen oyundan faydalanmıştır. İmitasyon oyunu bir sorgulayıcı ve biri kadın biri erkek olmak üzere iki sorgulanandan oluşmaktadır. Sorgulayıcı, sorgulananları görmemekte ve onlara sorduğu soruların cevapları yazılı olarak almaktadır. Sorgulayıcının amacı sorgulananlardan hangisinin kadın hangisinin erkek olduğunu bilmektir. Turing testinde sorgulayıcı bir insandır. Sorgulananlardan biri makine diğeri insandır. Eğer sorgulayıcı, sorguladıklarından hangisinin insan hangisinin makine olduğunu bilemezse, bu makine Turing testini geçmiş olmaktadır ve düşünebiliyor olarak kabul edilmektedir.

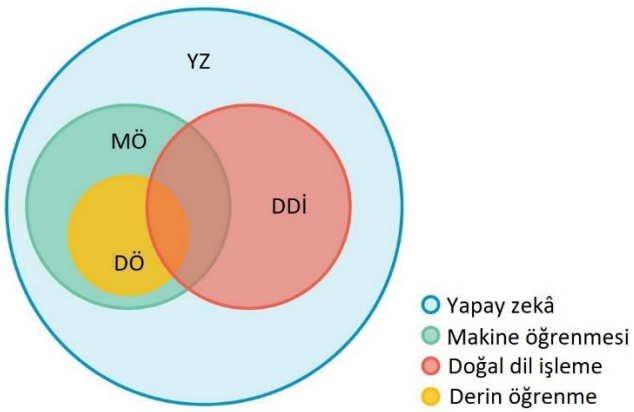
Yapay zekâ terimini ilk kez 1955 yılında yapay zekânın babası olarak kabul edilen John McCarthy önermiştir. 1956 yılında yapılan Dartmouth konferansı bu öneriyle birlikte ilk yapay zekâ konferansı olmuştur. Yine 1956 yılında Allen Newell ve Herbert Simon tarafından ilk çalışan yapay zekâ uygulaması olan "Logic Theorist" yazılmıştır. Bu program Russell ve Whitehead'in *Principia Mathematica* kitabının 2. bölümündeki teoremlerin çoğunu kanıtlamıştır. Simon, sayısal olmayan şekilde düşünebilen bir bilgisayar programı yazdıklarını söylemiştir. (Russell ve Norvig 2002)

Bu tarihlerden itibaren 1970lere kadar birçok yapay zekâ çalışması yapıldı. Fakat yapılan fonlamalara rağmen donanımsal yöndeki yetersizliklerin de etkisiyle bu çalışmalarda

istenen seviyelere ulaşılamadı. 1970-1990 yılları arasında yapay zekâya çalışmalarına yapılan fonlamalar azaldı ve bu yıllar "yapay zekâ kışı" olarak adlandırıldı. 1990lardan sonra tekrardan yapay zekâya karşı bir ilgi oluşmaya başladı. 1997 yılında IBM tarafından üretilen "Deep Blue", Garry Kasparov'u yenerek mevcut dünya satranç şampiyonunu yenebilen ilk bilgisayar oldu (Şekil 2.4).

Şekil 2.4. Garry Kasparov ve IBM Deep Blue (Ray 2018)

Günümüzde donanımların da güçlenmesiyle birlikte yapay zekâya olan ilgi artmış durumdadır. Birçok firma ve araştırmacı bu konularda çalışmalar yürütmektedir. Bu çalışmaların sonucunda yapay zekânın makine öğrenmesi, yapay sinir ağları, bilgisayarla görme, evrimsel algoritmalar, robotik, uzman sistemler gibi pek çok alt dalı ortaya çıkmıştır (Şekil 2.5).



Şekil 2.5. Yapay zekâ ve bazı alt dalları (Singh 2018'den değiştirilerek alınmıştır)

2.2. Makine Öğrenmesi

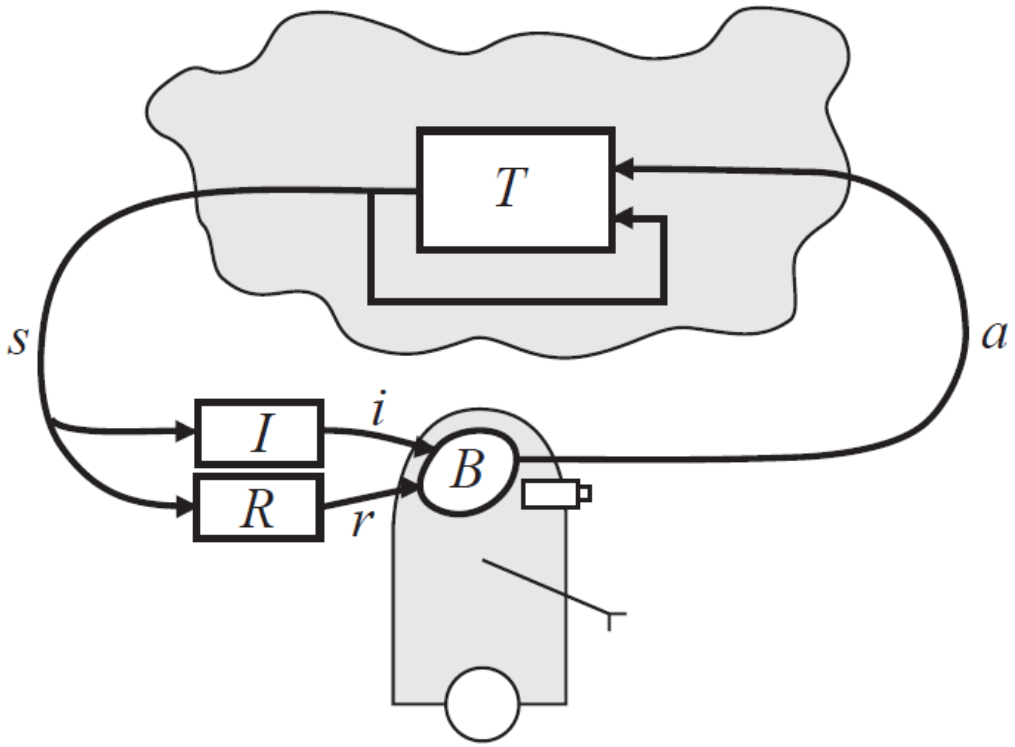
Makine öğrenmesi yapay zekânın bir alt dalıdır. Bilgisayarların, örnek veri veya geçmiş tecrübeleri kullanarak bir performans kriterini optimize edecek şekilde programlanmasıdır. Kişilerin doğrudan verilen problemi çözebilecek bir program yazamayıp örnek veri veya tecrübeye ihtiyaç duydukları durumlarda öğrenme gereklidir. İnsan uzmanlığının mevcut olmadığı ya da insanların uzmanlık konusunu ifade edemediği durumlarda da çözüm olarak öğrenme kullanılabilir (Alpaydın 2010). Makine öğrenmesinin birçok türü bulunmaktadır. Bunlardan bazıları danışmanlı öğrenme, danışmansız öğrenme ve pekiştirmeli öğrenmedir.

Danışmanlı öğrenmede kullanılan veri kümesindeki tüm veriler, beklenen çıktıları ile etiketlenmiş halde bulunmaktadır. Genellikle sınıflandırma ve regresyon problemlerini çözmek için kullanılmaktadırlar. Sınıflandırma problemlerinin eğitim aşamasında etiketli verileri girdi olarak alan sistem, matematiksel bir model oluşturur. Eğitim aşamasının ardından oluşturulan matematiksel model kullanılarak test aşamasında sisteme yeni gelen verilerin hangi sınıfa ait olduğu tahmin edilir. Regresyon problemlerinde ise çıktılar, girdinin durumuna göre belli bir aralıktaki herhangi bir sayısal değer olabilir.

Danışmansız öğrenmede eğitim verilerinde beklenen çıktılar bulunmamaktadır. Sadece girdi olarak kullanılacak verilerle eğitim yapılarak, verilerin içinde bir yapı bulunarak bunların gruplanması veya kümelenmesi sağlanmaktadır. Yeni gelen veriler için birtakım benzerlikler arayarak bu benzerliklerin bulunup bulunmamasına göre onları denk gelen gruplara atamaktadır.

Pekiştirmeli öğrenme; makine öğrenmesinin, bir ortamdaki yazılım ajanlarının kümülatif ödülü arttırmak için hangi adımları atmaları gerektiğiyle ilgilenen bir alt dalıdır. Basit bir pekiştirmeli öğrenme modeli Şekil 2.6'da verilmiştir. Bu modelde, ajan her iterasyonda ortamdan bir i girdisi ve s durumunu almaktadır. Bu girdilere göre bir a aksiyonu seçerek onu çıktı olarak üretmektedir. Çıktı, ortamın durumunu değiştirmektedir. Bu durum değişimi ajana skaler bir pekiştirme sinyali r olarak geri dönmektedir. Ajanın B davranışı, pekiştirme sinyalinden gelen değerlerin toplamlarını uzun vadede arttıracak aksiyonlar

seçmeye eğilimli hale gelmektedir. Sistem doğru tercihleri yapmayı zamanla deneme yanılma yöntemiyle öğrenmektedir (Kaelbling ve ark. 1996).



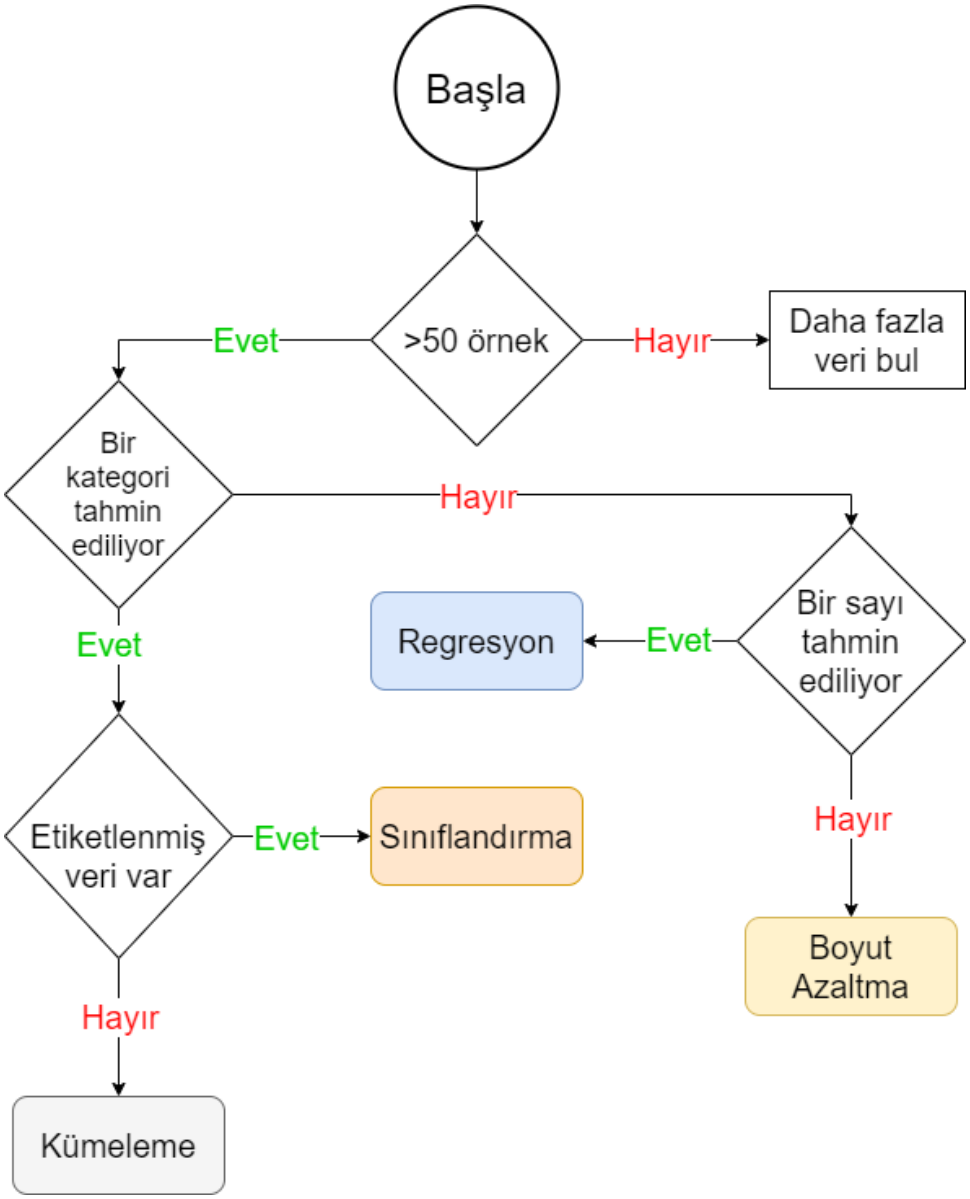
Şekil 2.6. Basit bir pekiştirmeli öğrenme modeli (Kaelbling ve ark. 1996)

Bu farklı makine öğrenmesi türlerini uygularken de pek çok farklı algoritma kullanılmaktadır. Bu algoritmalardan hangisinin tercih edileceğine, yapılacak işlemin türüne ve eğitim verisinin miktarına göre karar verilmektedir (Şekil 2.7). Bu algoritmalardan bazıları Naive Bayes, Destek Vektör Makinesi (Support Vector Machine, SVM), K-ortalamalar (K-means), Destek Vektör Sınıflandırıcı (Support Vector Classifier, SVC) algoritmalarıdır. Örneğin; eğer 100 bin örnekten az metinlerden oluşan bir veri kümesinde sınıflandırma problemi çözülecekse Naive Bayes algoritması kullanılabilecekken, aynı miktarda farklı türdeki veriler için SVM tercih edilebilmektedir.

2.2.1. Yapay sinir ağları

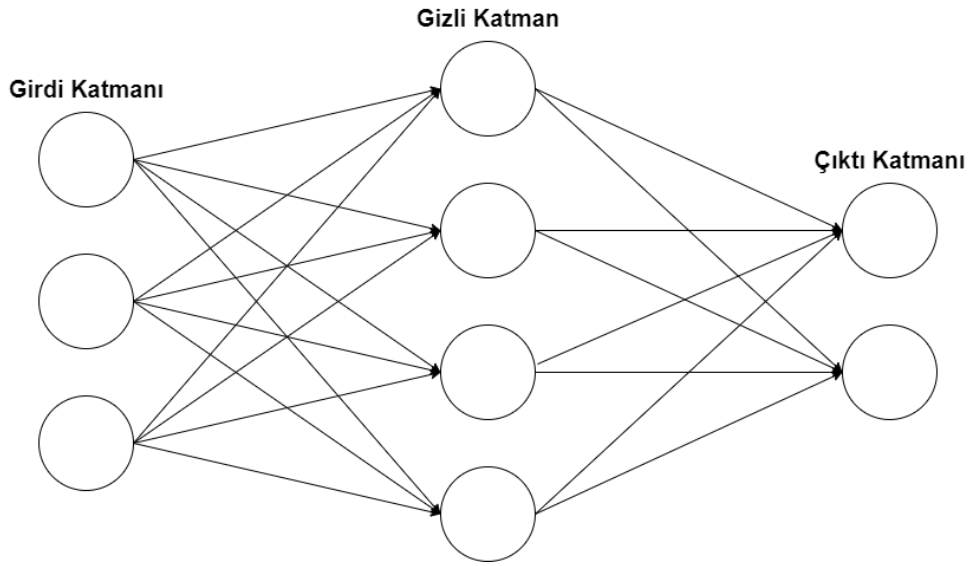
Bilgisayar bilimlerinde, yapay sinir ağlarının geçmişi yapay zekâyla hemen hemen aynıdır. McCulloch ve Pitts (1943) ve Hebb (1949) tarafından yapılan çalışmalarla temeli

atılmıştır. Yapay sinir ağları, birbirine bağlı nöron ya da düğüm olarak bilinen işleme elemanlarından oluşmaktadır. Her nöronun belli bir transfer fonksiyonunu işletip çıktı verdiği yönlü bir graftır. Çoğu zaman bu transfer fonksiyonu lineer olmayan bir fonksiyon olmaktadır (Yao 1999). Örüntü sınıflandırma, kümeleme, fonksiyon modelleme, tahmin yapma, optimizasyon, ilişkilendirme ve kontrol konularıyla ilgili problemlerin çözümünde diğer hesaplama araçlarından daha iyi sonuçlar elde etmektedirler (Basheer ve Hajmeer 2000).



Şekil 2.7. Scikit-learn kütüphanesinin algoritma tercih akış diyagramı (Anonim 2020d'den değiştirilerek alınmıştır)

Yapay sinir ağları genel olarak birkaç katmandan oluşmaktadır. Bir katmandaki nöronlar bir sonraki katmanın nöronlarına bağlanmaktadır. İlk katman girdi katmanı, son katman çıktı katmanı olarak adlandırılmakta ve bu katmanlar tüm yapay sinir ağlarında bulunmaktadır. Bu iki katman arasında farklı sayılarda katmanlar bulunabilmektedir. Bu katmanlar gizli katman olarak adlandırılır (Şekil 2.8). Katmanlar arasında, iki katmanın nöronlarını birbirine bağlayan bağlantılar vardır. Bu bağlantılar bir katmandaki nöronun çıktısını sonraki katmandaki nöronların girdisi olacak şekilde birbirine bağlamaktadır. Bu bağlantıların önemlerine göre farklı ağırlıkları vardır. Yapay sinir ağlarının eğitiminde amaç bu ağırlıkları, sistemi doğru tahminler yapacak hale getirecek şekilde güncellemektir.

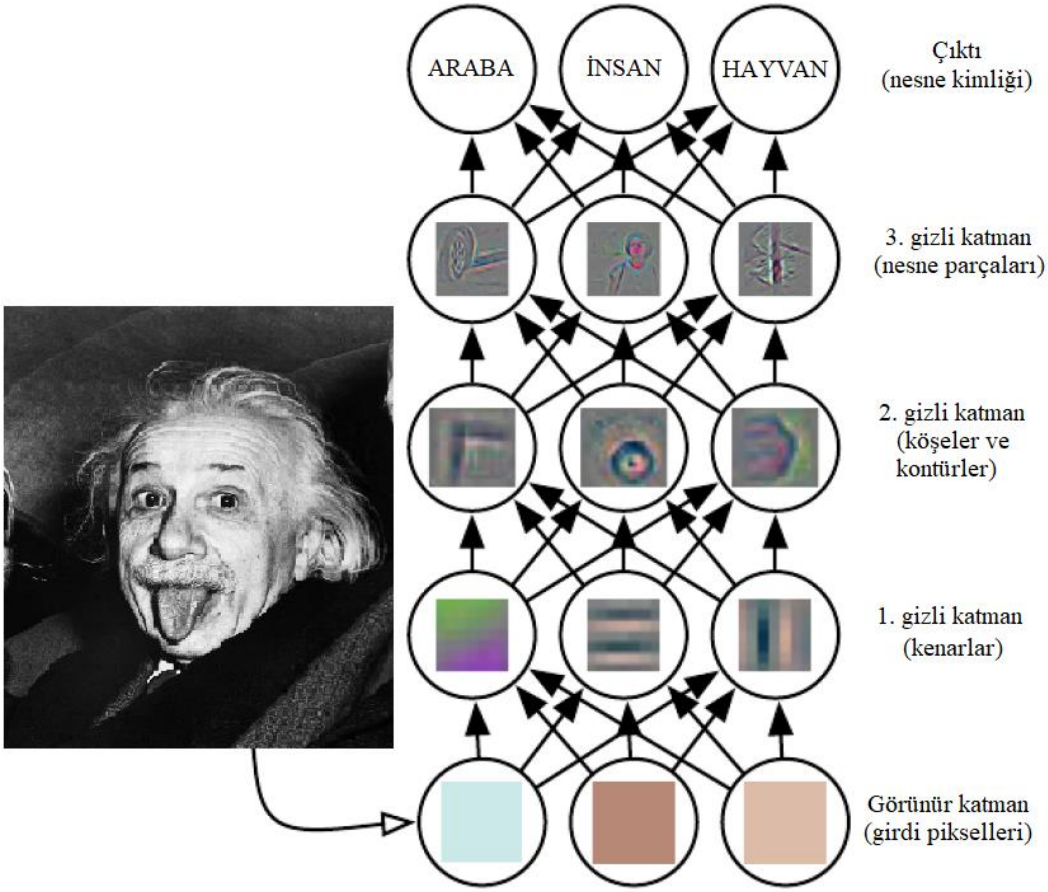


Şekil 2.8. Basit bir yapay sinir ağı yapısı

2.2.2. Derin öğrenme

Derin öğrenme, makine öğrenmesinin bir alt dalıdır. Genellikle konvolüsyonel (convolutional) yapay sinir ağlarını kullanmaktadır. Yapay zekânın ilk zamanlarında insanlar için çözmesi zor fakat formal ve matematiksel kurallarla ifadesi kolay problemler çözülmüştür. Fakat yapay zekâ için gerçek meydan okuma insanların çözmesi kolay ama formal ifade etmesi zor problemlerdir. Bu tarz problemlerin çözümü için problem hakkındaki bilginin doğrudan kodlanması yerine programın birtakım verilerden onlar hakkında kendi bilgisini elde etmesi gerekmektedir. Böylelikle makine öğrenmesi ortaya

çıkmıştır. Makine öğrenmesinde, veriden bilgiyi içeren özelliklerin seçilerek öğrenme algoritmalarını beslemesiyle birçok regresyon ve sınıflandırma problemi başarıyla çözülmüştür. Fakat bazı problemler için hangi özelliklerin seçilmesi gerektiğini bilmek çok zordur. Ham veriden hangi özelliklerin seçileceğini bulmak için özellik öğrenme kullanılmıştır. Bu yöntem veriden basit özelliklerin çıkarımında başarılı olmuştur. Derin öğrenme ile bu basit özellikler bir araya getirilmiş ve Şekil 2.9'da gösterildiği gibi bilgisayarların bu basit özelliklerden daha kompleks konseptleri oluşturması sağlanmıştır (Goodfellow ve ark. 2016).



Şekil 2.9. Derin öğrenmede karmaşık özelliklerin çıkarımı (Goodfellow ve ark. 2016'dan değiştirilerek alınmıştır)

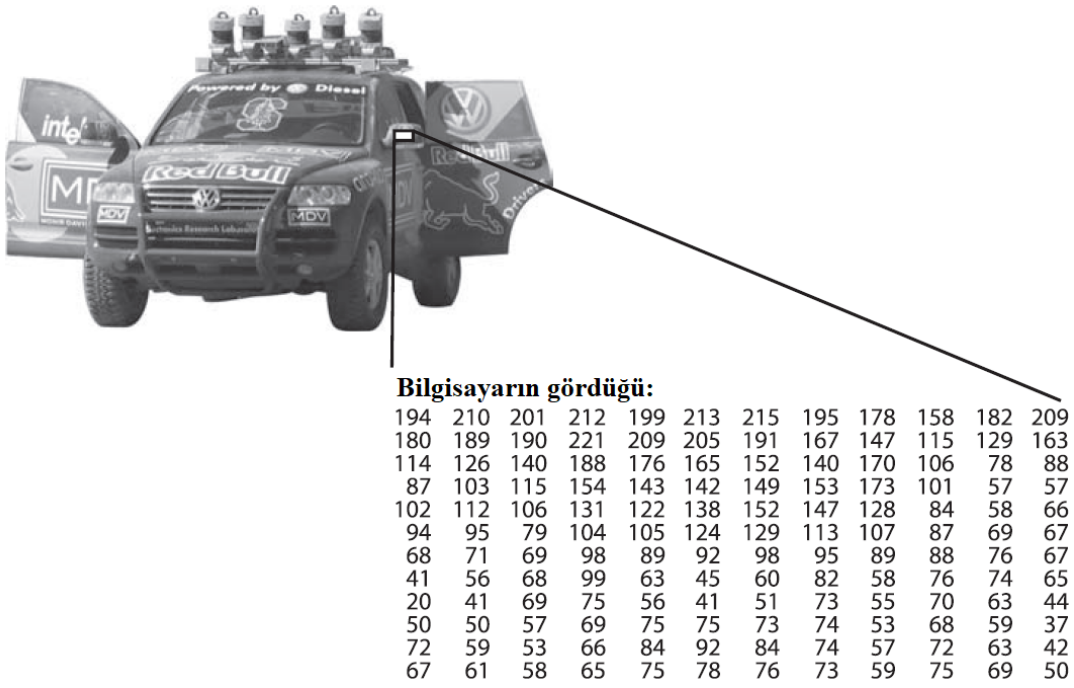
Derin öğrenme uygulamaları artık günlük hayatta birçok yerde karşımıza çıkmaktadır. İnternet üzerinde yaptığımız aramalardan, sosyal medyada karşımıza çıkan içeriklere ve e-ticaret sitelerinin yaptığı önerilere kadar günlük hayatta karşılaştığımız çoğu şeyin arkasında derin öğrenme yatmaktadır (LeCun ve ark. 2015). Bunlar dışında görsellerin

içindeki nesnelerin tanınmasında, görselin içeriğinin yazıya dökülmesinde kullanılmaktadır. Doğal dil işlemede, konuşma tanıma ve kurulan cümleyi anlama konusunda uygulamaları bulunmaktadır. Google'ın çeviri servisinde derin öğrenmeden faydalanılmaktadır (Turovsky 2016). Sanatta, verilen tablonun hangi ressama ait olduğunu tahmin edebilen (Smith ve Leymarie 2017) ya da verilen bir tablodaki stili başka herhangi bir resme aktarabilen sistemler (Şekil 2.10) derin öğrenme yardımıyla yapılmaktadır (Gatys ve ark. 2016). Tıpta, kanser hücrelerinin sınıflandırılması ve lezyon tespiti gibi medikal görsellerin analizinde kullanılmaktadır (Litjens ve ark. 2017). Ayrıca ilaç keşfi için de derin öğrenmenin kullanıldığı durumlar bulunmaktadır. Bu uygulamaların birçoğu danışmanlı öğrenme yöntemiyle yapılmaktadır. Uzun vadede, insan ve hayvan öğrenmesine daha yakın bir yöntem olan danışmansız öğrenmenin bu alanda çok daha önemli olacağı tahmin edilmektedir (LeCun ve ark. 2015).

Şekil 2.10. Sanatsal stil transferi örnekleri **A)** Andreas Praefcke tarafından çekilen farklı stillere aktarılacak olan fotoğraf **B)** J.M.W. Turner'ın *The Shipwreck of the Minotaur* tablosu **C)** Vincent van Gogh'un *The Starry Night* tablosu **D)** Edvard Munch'ın *Der Schrei* tablosu (Gatys ve ark. 2016)

2.3. Bilgisayarla Görme

İnsanlar için etraflarına baktıkları zaman üç boyutlu bir ortamı ve bu ortamda bulunan nesneleri algılamak çok kolay ve çoğu zaman farkında olunmadan gerçekleşen bir işlemdir. Fakat bilgisayarlar için görüntülerden anlam çıkarmak bu kadar kolay değildir. Bu görevi başarmak için yapılan çalışmalar bilgisayarla görme alanında toplanmaktadır. Bilgisayarla görme, fotoğraf veya video verisinin, bir karar ya da farklı bir gösterime dönüştürülmesi işlemidir (Bradski ve Kaehler 2008). Bilgisayarlar, görüntüleri birtakım sayılar örgüsü olarak görmektedir (Şekil 2.11). Bu sayılardan anlam çıkarmak için bazı görüntü işleme ya da yapay zekâ tekniklerini kullanmak gerekmektedir.

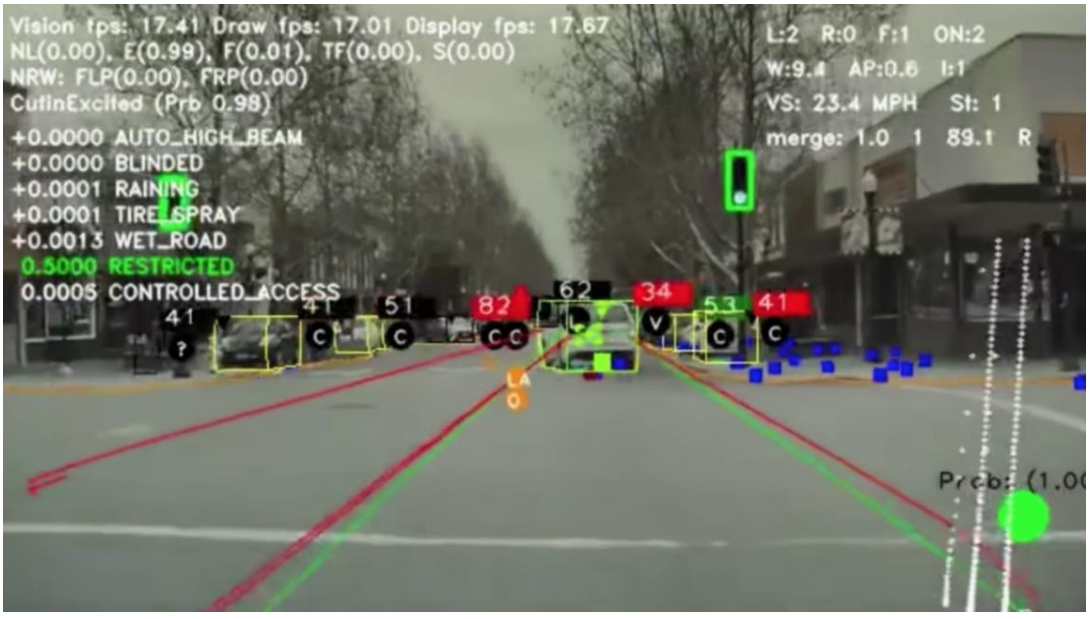


Şekil 2.11. Bir görselin bilgisayar tarafından okunması (Bradski ve Kaehler 2008'den değiştirilerek alınmıştır)

Yapılmak istenen işe göre uygulanabilecek pek çok teknik bulunmaktadır. Bunlardan bazıları piksel dönüşümü, renk dönüşümü, histogram eşitleme gibi nokta işlemleridir (Szeliski 2011). Ayrıca piksel komşulukları kullanılarak konvolüsyonel filtreleme ile gürültü giderme ya da yumuşatma işlemleri yapılabilmektedir. Fourier dönüşümleri yapılarak görüntüler uzamsal alandan frekans alanına aktarılabilmekte ve böylece bazı

filtreler üzerlerinde daha kolay uygulanabilmektedir (Forsyth ve Ponce 2012). Lineer filtreler ile kenarlar ve çizgiler algılanabilmektedir. Çizgileri algılamak için Hough dönüşümü gibi yöntemlerden faydalanılabilmektedir. Kümeleme ile segmentasyon yapılabilmekte, görsellerin boyutlarını düşürmek için kuantalama işlemleri kullanılabilmektedir. Harris köşe algılayıcı ile köşeler bulunabilmektedir. Ölçek Değişimsiz Özellik Dönüşümü (Scale Invariant Feature Transformation, SIFT) ve HOG gibi özellik çıkarıcılar ile görsel içindeki özellikler bulunup, aynı görüntüyü farklı açılardan içeren görüntüler arasındaki bağlantılar tespit edilebilmektedir. Görsel içinde bulunan nesnelerin tespiti için çeşitli makine öğrenmesi tekniklerinden faydalanabilmektedir.

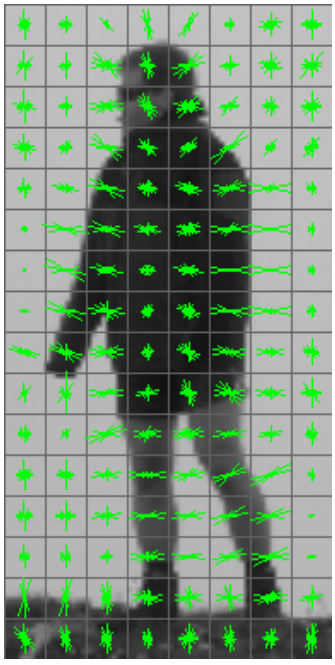
Bilgisayarla görmenin günümüzde hızla artan sayıda uygulama alanı bulunmaktadır. Tıpta ultrason ve X-ray gibi medikal görüntülerden anlamlar çıkarmak ve teşhis koymak için kullanılmaktadır. Otonom arabalar, insansız hava araçları ve gezegen keşif araçları gibi otonom araçlar için kilit konumdadır (Şekil 2.12). Askeri alanda füze yönlendirme ve hedef tespiti gibi konularda, güvenlik alanında güvenlik kameralarındaki görüntülerden olağanüstü durum tespiti yapmada kullanılmaktadır.



Şekil 2.12. Tesla'nın otomatik pilot sisteminden bir görüntü (Hart 2020)

2.4. Nesne Tanıma

Nesne tanıma, bilgisayarla görme ve yapay zekâ konularının ortak bir alt dalıdır. Bir görüntünün içindeki bir veya daha fazla nesnenin varlığını tespit etmeyi ve konumlarını belirlemeyi amaçlar. Nesne tanıma probleminin çözümünde kullanılan en temel yöntemlerden biri, arka plan çıkarma yöntemiyle araçların tanınmasıdır. Gupte ve ark. (2002) tarafından yapılan çalışmada, yol görüntülerindeki hareketli bölgeler, arka plandan ayrılarak bu nesneler araç olarak incelenmiştir ve 20 dakikalık bir videoda araçların %90'ı doğru bir şekilde algılanmış, %70'i de doğru sınıflandırılmıştır. Nesne tanıma probleminin çözümünde derin öğrenmenin ilk kullanımı ise LeCun ve ark. (1998) tarafından tanıtılan LeNet yöntemi ile karşımıza çıkmıştır. Bu yöntemle el yazısı rakamlar %82'lik bir doğrulukla sınıflandırılmış ve postanelerdeki iş yükünde büyük bir azalma ve süreçlerde hızlanma sağlanmıştır. Dalal ve Triggs (2005) tarafından, güvenlik kameraları ve benzer görüntülerdeki insanları bulup görüntüdeki insanlarla ilgili çalışmalar yapmak adına HOG ile insan tanıma çalışması yapılmıştır (Şekil 2.13). Bu çalışmada insan görsellerinin türevleri alınıp, bu türevlerin genlik ve yönlerinin vektörleri oluşturulmuş ve destek vektör makinesi tipinde bir sınıflandırıcı görüntülerdeki insanları bulmak üzere eğitilmiş ve %89 oranında başarı yakalamıştır.



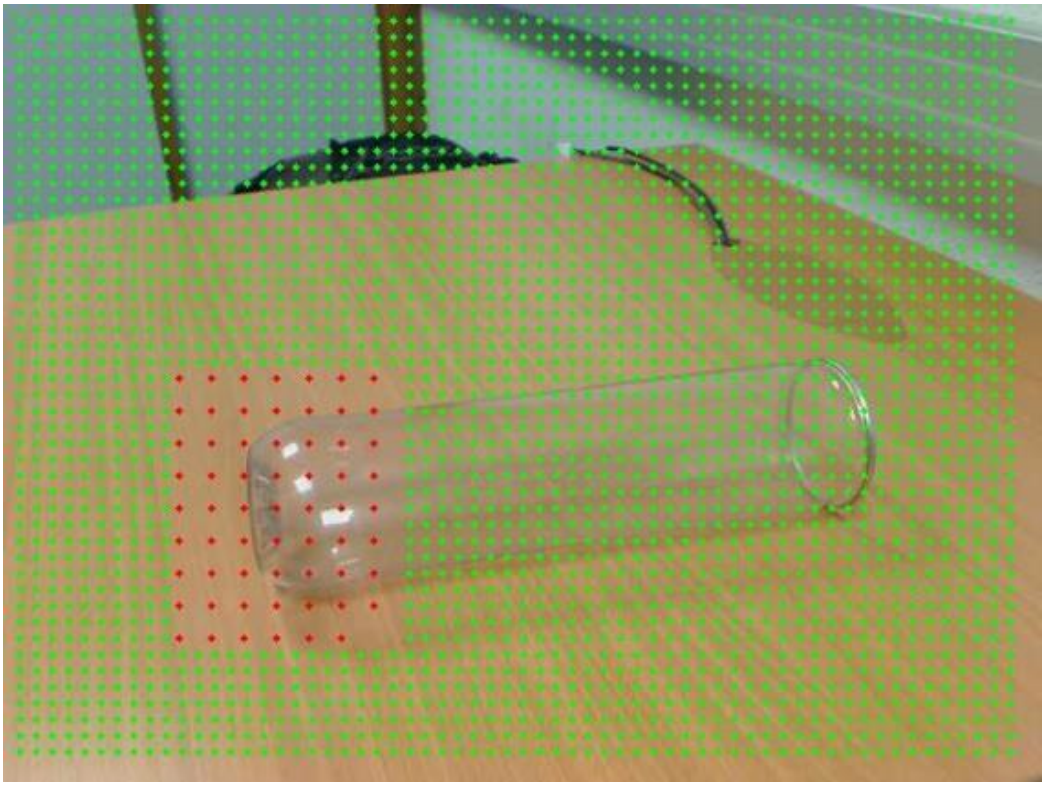
Şekil 2.13. İnsan görselinden çıkarılan HOG özellikleri (Bertok 2016)

Krizhevsky ve ark. (2012) tarafından ImageNet Büyük Ölçekli Görsel Tanıma Yarışması (ImageNet Large Scale Visual Recognition Challenge, ILSVRC) (Russakovsky ve ark. 2015) için geliştirilen AlexNet yöntemi ile derin öğrenme kullanılarak nesne tanıma probleminde büyük yol kat edilmiştir. 1000 sınıf halinde bulunan nesneleri sınıflandırma gibi görevler içeren bu yarışmada daha önceki yöntemlere kıyasla başarılı sonuçlar elde edilmiştir. He ve ark. (2016) tarafından sunulan ve ILSVRC-2015 yarışmasını kazanan bir diğer derin öğrenme yöntemi olan ResNet sayesinde ağların daha da derinleşmesi sağlanmış ve daha başarılı sonuçlar elde edilmiştir. Sabour ve ark. (2017) tarafından yapılan çalışmada kullanılan CapsNet yöntemi, el yazısı rakam tanıma özelinde, derin öğrenmenin sahip olduğu birtakım eksiklikleri gidermiş ve oldukça başarılı sonuçlar elde etmiştir.

Şeffaf nesne tanıma adına pek çok yöntem ve farklı donanımlar kullanılmıştır. Kullanılan yöntemlerden biri kolektif ödül tabanlı yaklaşımdır. Kompella ve Sturm (2011) tarafından tanıtılan bu yöntemde görüntünün içinden bir parça seçilerek, bu parçanın içindeki ve dışındaki noktaların birbirleriyle olan ilişkisi incelenmiştir (Şekil 2.14). Bu ilişkilerden elde edilen değerler kullanılarak kolektif ödül ve sınıflandırma sürecinin sonucunda seçilen bölgenin şeffaf bir nesneye karşılık gelip gelmediği tespit edilmiş ve 50 örneklik bir test kümesinde %77,19'luk bir başarı elde edilmiştir.

Klank ve ark. (2011) tarafından yapılan çalışmada kızıl ötesi ışık kullanılarak şeffaf nesnelerin tespiti sağlanmıştır. Bu çalışmada cam gibi şeffaf nesnelerin belli dalga boylarındaki ışığı soğurma özelliği kullanılmıştır. Bu yöntemi uygulamak için bir adet kızıl ötesi ışık kaynağı ve özel bir kamera gerekmektedir. Tanımayı kolaylaştırmak için stereo görüntü kullanılmıştır. Kameradan veri okunarak yoğun bölgeler aday olarak çıkarılır. Adaylar sisteme, mesafe ve yoğunluk haritaları ve nokta bulutu ile geri döner. Aktüatör ve sensörlerin üzerinde üzerinde bulunduğu robot bir miktar hareket ettirilerek stereo görüntü elde edilir. Şeffaf nesneler bu görüntüde ışığın arka plandan yansıması sebebiyle gölge gibi görünür. Bu yöntem kullanılarak hem şeffaf hem şeffaf olmayan nesnelerin sınıflandırması yapılmıştır. 44 şeffaf olmayan nesne %100 doğrulukla etiketlenirken, 105 şeffaf nesnenin %55,24'ü şeffaf olarak etiketlenmiştir. Şeffaf olmayan nesneleri tanımada bugün en çok kullanılan yöntemler istatistiksel öğrenme

yöntemleridir. Şeffaf nesne tanıma probleminin çözümünde de Khaing ve Masayuki (2018) tarafından yapılan çalışmada konvolüsyonel sinir ağları kullanılmıştır. Bu çalışmada yapılan deneyler sonucunda %85,1'lik bir Kesinlik (Precision, P) değeri elde edilmiştir.



Şekil 2.14. Seçilen bölgenin, karşılaştırılacak iç ve dış noktaları (Kompella ve Sturm 2011)

3. MATERYAL ve YÖNTEM

Bu bölümde, çalışmada kullanılan veri kümesi ve dört farklı yöntem ile ilgili bilgiler sunulacaktır. Çalışmada kullanılan yöntemlerden ilki olan LeNet, derin öğrenmenin kabul edilen ilk modelidir. İkinci kullanılan yöntem olan AlexNet ise derin öğrenmeyi, ilk ortaya çıkışından uzun bir süre sonra tekrar popüler hale getirmiştir. Üçüncü kullanılan yöntem, derin öğrenmedeki ağların derinleştikçe karşılaştığı problemlere çözümler bularak daha derin yapıların kurulmasını sağlayan ResNet'tir. Kullanılan son yöntem ise nesne tanıma problemlerine yeni bir çözüm getiren ve bizim de üzerine yoğunlaştığımız CapsNet'tir.

3.1. Veri Kümesi

Bu çalışmada ImageNet (Deng ve ark. 2009) üzerinden elde edilen ve içerisinde şeffaf olan ve olmayan bardakların görselleri bulunan veri kümesi kullanılmıştır. Veri kümesindeki şeffaf bardakların sayısı 2707, şeffaf olmayan bardakların sayısı 3292'dir. Şeffaf olan veriler ImageNet'in içeriğindeki bira bardağı, kahve kupası, kupa, çay fincanı ve su bardağı sınıflarındaki şeffaf olan bardaklardan elde edilmiştir. Şeffaf olmayan veriler ise su bardağı sınıfı hariç şeffaf olanlarla aynı sınıflardaki şeffaf olmayan verilerden elde edilmiştir. Hangi tür bardakların, hangi sınıfta kaç adet bulunduğu Çizelge 3.1'de verilmiştir.

Çizelge 3.1. Bardak türlerinin sınıflara göre dağılımı

	Bira Bardağı	**Kahve Kupası**	**Kupa**	**Çay Fincanı**	**Su Bardağı**
Şeffaf	1297	41	254	75	1040
Şeffaf Olmayan	1	1164	1044	1083	0

Bu verilerin tamamı içeriğinde sadece bardağı içerecek kare şeklinde görseller olacak şekilde elle (manual) kırpılarak oluşturulmuştur. Görseller 3 kanallı Kırmızı-Yeşil-Mavi (Red-Green-Blue, RGB) olarak kullanılmıştır. Veriler ilk olarak 32x32 piksel boyutuna

getirilmiştir. Daha sonra ağların özelliklerine göre her biri yeniden boyutlandırılmıştır. Şeffaf ve şeffaf olmayan bardak resimlerine ait örnekler Şekil 3.1'de sunulmuştur.

Şekil 3.1. Şeffaf ve şeffaf olmayan bardak örnekleri

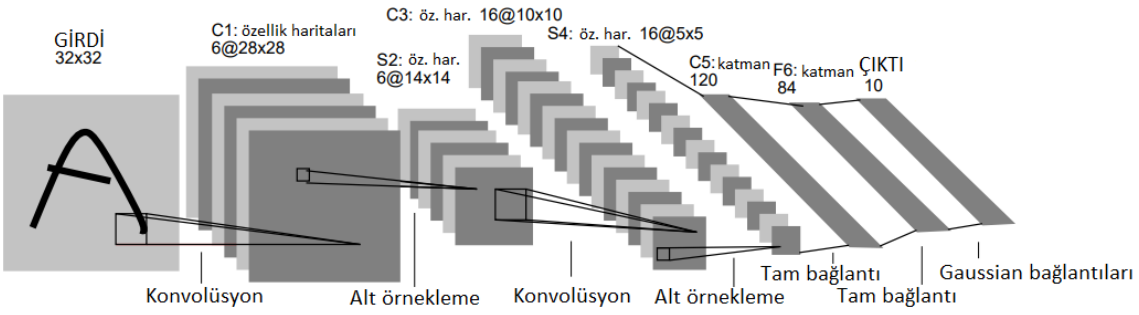
3.2. Yöntemler

Çalışmada kullanılan yöntemlerin giriş katmanları farklı boyutlarda veriler kabul ettiği için bazılarının giriş katmanları, bazıları için ise giriş katmanını besleyen verilerde değişiklik yapılmıştır. Ayrıca farklı problemlerin çözümü için üretilmiş olan bu yöntemlerin çıkış katmanları da bu çalışmadaki sınıf sayısından farklı sayıda sınıfa sahiptir. Bu yüzden tüm modellerin çıkış katmanları iki sınıflı bir sınıflandırma yapacak şekilde değiştirilmiştir. Bazı modellerde, bu değişimler ara katmanlarda da birtakım değişikliklere sebep olmuştur.

3.2.1. LeNet

LeNet modeli, LeCun ve ark. (1998) tarafından önerilmiştir ve ilk başarılı konvolüsyonel sinir ağı modeli olarak kabul edilmektedir. Geliştirilme amacı el yazısı rakamları tanımaktır. Yapısında bulunan ortaklama (pooling) katmanında, derin öğrenmedeki genel yapının aksine, maksimum değil ortalama değerleri kullanmaktadır. LeNet'in genel yapısı Şekil 3.2'de gösterilmiştir.

Bu çalışmadan kullanılan LeNet-5 modeli 7 katmandan oluşmaktadır. Bunlardan üçü konvolüsyon (convolution) katmanı, ikisi alt örnekleme (subsampling) katmanı, biri Tam Bağlantılı (Fully Connected, TB) katman ve biri çıkış katmanıdır. LeNet-5 modelinde girdi katmanı 32x32 piksel boyutunda üç kanallı RGB bir görsel almaktadır. İlk konvolüsyon katmanına geçilirken 5x5 boyutunda 6 kanallı bir filtre (kernel) birer birer atlama (stride) yapılarak bu 32x32x3 boyutundaki girdi, 28x28x6 boyutuna dönüşür. Daha sonra 2x2 boyutunda bir filtre ile her seferinde 2 adım atlama ile alt örnekleme yapılarak boyut 14x14x6 haline gelir. 16 kanallı 5x5 boyutunda bir filtre ile bir piksel atlama yapılarak tekrar konvolüsyon yapılır ve 10x10 boyutundaki 16 kanallı katman elde edilir. Bu katmanda tekrar 2x2 boyutunda 2 kaydırmalı filtreyle alt örnekleme yapılarak sonraki katman 5x5x16 boyutuna getirilir.



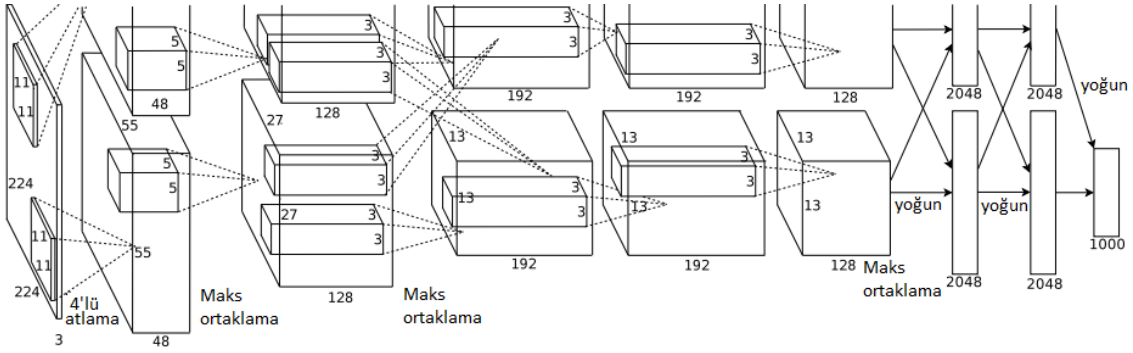
Şekil 3.2. LeNet modelinin genel yapısı (LeCun ve ark. 1998'den değiştirilerek alınmıştır)

Elde edilen katman yassılaştırılarak 120 özellikli tam bağlantılı katmana bağlanır. Bu katmandan da 84 özellikli katmana bağlantı yapılır. Son olarak bu 84 özellik çıktı katmanına bağlanır. Modelin ilk hali rakam tanıma amaçlı olduğu için 10 sınıflı bir veriyi

sınıflandıracak şekilde oluşturulmuştur. Bu çalışmada bu yapı bardakların şeffaf olup olmadığını belirlemek için 2 sınıfı sınıflandıracak hale getirilmiştir. Girdi ve çıktı katmanları hariç diğer tüm katmanlar aynı boyutlarda kullanılmıştır. Optimizasyon yöntemi olarak Adam optimizer (Kingma ve Ba 2014) ve kayıp fonksiyonu olarak kategorik çapraz entropi (categorical cross-entropy) (Nasr ve ark. 2002, Anonim 2019a) kullanılmıştır.

3.2.2. AlexNet

Krizhevsky ve ark. (2012) tarafından önerilen AlexNet modeli, LeNet'in yayınlanmasından 14 yıl sonra derin öğrenmenin tekrar popüler hale gelmesini sağlayan çalışmadır. Aktivasyon fonksiyonu olarak Doğrultulmuş Lineer Ünite (Rectified Linear Unit, ReLU) (Nair ve Hinton 2010), ortaklama katmanı olarak maksimum ortaklama kullanır. Aşırı uyum göstermeyi önlemek için bırakma (dropout) katmanları kullanır. Bu katmanlarda önceki ve sonraki katmanlar arasındaki bağlar her veri için rastgele koparılarak eğitim yapılır. AlexNet'in genel yapısı Şekil 3.3'te gösterilmiştir.



Şekil 3.3. AlexNet modelinin genel yapısı (Krizhevsky ve ark. 2012'den değiştirilerek alınmıştır)

AlexNet, 5 konvolüsyonel ve 3 tam bağlı katmandan oluşmaktadır. Tüm katmanlardan sonra aktivasyon fonksiyonu olarak ReLU (Nair ve Hinton 2010) kullanılmaktadır. Giriş katmanında 3 kanallı 224x244 boyutunda veriler kabul etmektedir. İlk katmanda bu verilere 11x11 boyutunda, 96 kanallı bir filtre 4 piksel atlama yapılarak uygulanmaktadır. Böylece bir sonraki katmanda 54x54x96 boyutunda bir veri elde edilmektedir. Elde edilen katmana 3x3 boyutunda 2 atlama ile maksimum ortaklama yapılmaktadır. Bu işlem

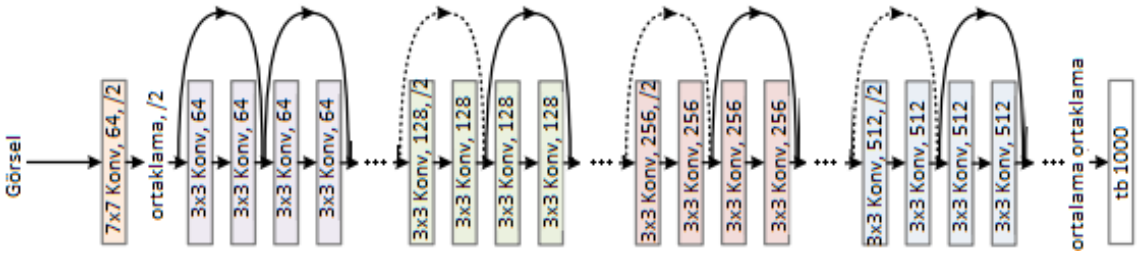
sonucunda katmanın boyutu 26x26x96 haline gelmektedir. İkinci konvolüsyon katmanında 5x5 boyutunda 256 kanallı bir filtre 1 atlama ile uygulanmaktadır. Böylece katmandaki veriler 26x26x256 boyutuna gelmektedir. Bu katmana 3x3 boyutunda 2 atlamalı filtre ile maksimum ortaklama yapılmaktadır. Bu işlem sonucunda boyut 12x12x256'ya düşmektedir.

Üçüncü ve dördüncü konvolüsyon katmanlarında 384 kanallı bir filtre kullanılmaktadır. Bu filtrenin boyutu 3x3 atlaması 1'dir. Bu katmanlardan sonra ortaklama işlemi yapılmamaktadır. İşlem sonunda boyut 12x12x384 haline gelmektedir. Beşinci konvolüsyon katmanında 256 kanallı 3x3 boyutunda 1 atlamalı bir filtre uygulanmakta ve veri 12x12x256 boyutuna gelmektedir. Daha sonra 3x3 boyutunda 2 atlamalı filtre ile maksimum ortaklama yapılarak 5x5x256 boyutunda veri elde edilmektedir.

Elde edilen katman yassılaştırılarak 6400 parametreli bir veri elde edilmektedir. İlk iki tam bağlı katmanda bu parametre sayısı 4096'ya indirilmektedir. Üçüncü tam bağlı katmanda parametre sayısı 1000 yapılarak 2 elemanlı çıkış katmanına bağlanmaktadır. Tam bağlı katmanların bulunduğu bölümde her katman arasında bırakma katmanları bulunmaktadır. Bu katmanların özelliği, tam bağlı katman arasındaki bağlantıları her döngüde (epoch) rastgele koparark aşırı öğrenmeyi önlemektir. Optimizasyon yöntemi olarak Adam optimizer (Kingma ve Ba 2014) ve kayıp fonksiyonu olarak ikili çapraz entropi (binary cross-entropy) (Anonim 2019b) kullanılmıştır.

3.2.3. ResNet

He ve ark. (2016) tarafından tanıtılan ResNet modeli, daha hızlı eğitilebilen bir ağ yapısı sunmaktadır. Yapay sinir ağlarında, ağın derinliği arttıkça, geri yayılımlı eğitimde aktivasyon fonksiyonlarının ürettiği değerler sıfıra yaklaşmaktadır. Bu problem derin öğrenmeyi, adında geçen derinlikten uzaklaştırmaktadır. Bu model, bu problemi çözmek için iki katman önceki aktivasyon katmanı çıktısını, iki katman sonraki aktivasyon katmanının girişine eklemektedir. Böylece geniş bir ağ yapısı yerine, derin bir ağ yapısı oluşturmuş olur. 152 katmanlı yapısıyla, katıldığı ILSVRC'de o zamana kadarki en derin ağ yapısı olmuştur. ResNet'in örnek bir genel yapısı Şekil 3.4'te gösterilmektedir.

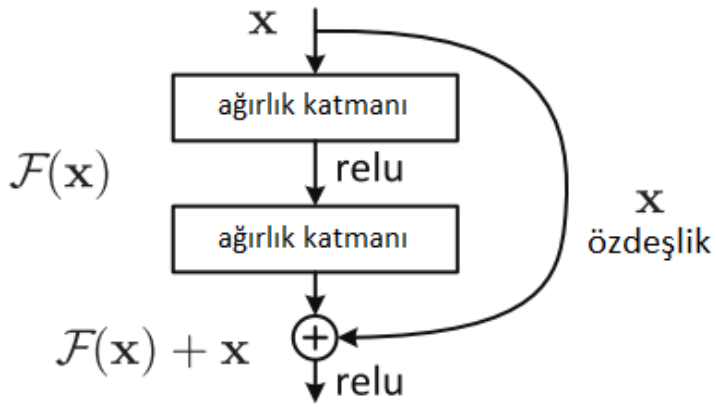


Şekil 3.4. ResNet modelinin genel yapısı (He ve ark. 2016'dan değiştirilerek alınmıştır)

ResNet, derin ağlarda görülen gradyan yok olması (gradient vanishing) problemini çözmüştür. Yapay sinir ağlarına katman ekledikçe bir sonraki katmanın öğrenme hızı artmaktadır. Daha fazla katman eklemek ağın daha karmaşık fonksiyonları öğrenmesini sağlamakta ve tahminlerdeki doğruluğu arttırmaktadır. Fakat geri yayılım sırasında ağırlıklarla ilgili kayıp gradyanlarını hesaplarken, ağda geriye gittikçe gradyanlar küçülmeye meyillidir. Bu durum, önceki katmanlardaki nöronların sonraki katmanlardaki nöronlara göre çok yavaş öğrenmesi anlamına gelmektedir. İlk katmanların görevi basit örüntüleri öğrenmek ve algılamaktır. Bu katmanların öğrenmesi yavaşladığında ve temel örüntüler doğru bir şekilde tespit edilemediğinde tüm ağın başarısı düşmektedir. Bu yüzden ilk katmanların öğrenmedeki başarısı ağın başarısında kritik bir rol oynamaktadır.

ResNet'te bu problem atlama bağlantısı (skip connection) kullanılarak aşılmıştır. Bu bağlantıları uygulamak için artık (residual) bloklar kullanılmıştır. Bu bloklar içerisinde klasik derin öğrenme modellerinin katmanlarında bulunan konvolüsyon, aktivasyon gibi işlemleri bulundurur. Birkaç blokta bir, birkaç önceki bloğun aktivasyon çıktısı, mevcut bloğun aktivasyonunun girdisine eklenir. Böylece gradyanların zamanla yok olmasının önüne geçilmiştir. Şekil 3.5'te örnek bir artık blok gösterilmiştir. Bu çalışmada kullanılan ResNet-50 modeli, 50 katmandan oluşmaktadır. Her ağırlık katmanı sırasıyla konvolüsyon, yığın normalleştirme (batch normalization) ve aktivasyon işlemleri bulunmaktadır. Her üç katmanda bir, üç önceki katmanın aktivasyon çıktısı, mevcut katmanın aktivasyon girdisine eklenmektedir. Son aktivasyonun çıktısı ortalama ortaklama katmanına bağlanmaktadır. Bu katmanın çıktıları, çıkış katmanının girdisi olmaktadır. En sonda iki sınıflı bir çıkış vererek nesnenin şeffaf olup olmadığını sınıflandırmaktadır. Tıpkı AlexNet'te olduğu gibi, optimizasyon yöntemi olarak Adam

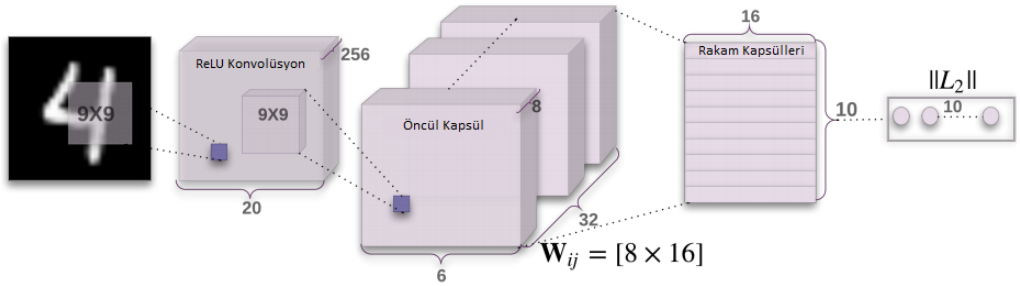
optimizer (Kingma ve Ba 2014) ve kayıp fonksiyonu olarak ikili çapraz entropi (Anonim 2019b) kullanılmıştır.



Şekil 3.5. Artık blok örneği (He ve ark. 2016'dan değiştirilerek alınmıştır)

3.2.4. CapsNet

Sabour ve ark. (2017) tarafından tanıtılan CapsNet, LeNet'ten bu yana gelişmekte olan derin öğrenme çözümlerine yeni bir boyut kazandırmıştır. CapsNet'in genel yapısı Şekil 3.6'da gösterilmiştir. Bu yeni modelin testleri de tıpkı LeNet gibi MNIST veri kümesi (LeCun ve Cortes 2010) üzerinde yapılmıştır. Derin öğrenme konusunda önemli çalışmalara sahip kişiler tarafından daha etkili yeni bir yöntem olarak sunulmuştur. CapsNet; tıp (Afshar ve ark. 2018, Mobiny ve Van Nguyen 2018), doğal dil işleme (Bae ve Kim 2018, Wang ve ark. 2018) ve tarım (Li ve ark. 2019) gibi çeşitli alanlarda kullanılmıştır.



Şekil 3.6. CapsNet modelinin genel yapısı (Sabour ve ark. 2017'den değiştirilerek alınmıştır)

Bu çalışmadan önce CapsNet'in işleyişini daha iyi anlayabilmek için bir ön çalışma yapılmıştır (Bilgin ve Mutludoğan 2019). Bu çalışmada Amerikan alfabesindeki harflerin işaret dili ile temsilleri eğitim verisi olarak kullanılmıştır. Bu veri kümesi, işaret dili temsili hareket gerektiren J ve Z harfleri hariç İngilizce alfabedeki harfleri temsil eden 24 sınıftan oluşmaktadır. Her sınıf için örnek veri Şekil 3.7'de gösterilmiştir. Eğitim kümesinde 27455, test kümesinde 7172 örnek veri bulunmaktadır. Veri kümesindeki veri sayısının sınıflara göre dağılımı Çizelge 3.2'de verilmiştir.

Şekil 3.7. Ön çalışmada kullanılan veri kümesinin sınıflarından örnekler (Bilgin ve Mutludoğan 2019)

Çizelge 3.2. Ön çalışmadaki verilerin sınıflara göre dağılımı (Bilgin ve Mutludoğan 2019)

Sınıf	A	B	C	D	E	F
Adet	1126	1010	1144	1196	957	1204
Sınıf	G	H	I	K	L	M
Adet	1090	1013	1162	1114	1241	1055
Sınıf	N	O	P	Q	R	S
Adet	1151	1196	1088	1279	1294	1199
Sınıf	T	U	V	W	X	Y
Adet	1186	1161	1082	1225	1164	1118

Yapılan ön çalışmada, CapsNet'in yanı sıra aynı veri kümesi kullanılarak LeNet-5 modeli de eğitilmiştir. Bu çalışmada LeNet %82'lik bir sınıflandırma doğrulaması elde ederken, CapsNet ilk aşamada %88 sınıflandırma doğruluğuna ulaşmıştır. Daha sonra veri arttırma metotları ile yeni veriler oluşturularak CapsNet eğitildiğinde sınıflandırma doğruluğu %95'lere ulaşarak CapsNet modelinin görsel sınıflandırma konusundaki başarısı görülmüştür. Tüm deneylere ait Doğruluk (Accuracy, A), Kesinlik, Duyarlılık (Recall, R) ve F-Skoru (F-Score, FS) metrikleri Çizelge 3.3'te verilmiştir.

Çizelge 3.3. Ön çalışmanın deney sonuçları (Bilgin ve Mutludoğan 2019)

Modeller	**Metrikler**			
	Doğruluk	**Kesinlik**	**Duyarlılık**	**F-Skoru**
LeNet	%82,19	%81,24	%81,82	%80,95
CapsNet	%88,93	%84,48	%89,04	%86,41
CapsNet Arttırılmış	%95,08	%91,11	%95,63	%93,22

Konvolüsyonel sinir ağları her ne kadar başarılı sonuçlara sahip olup derin öğrenmenin bugünkü popülerliğini sağlamış olsa da birtakım eksikliklere sahiptir. Örneğin; konvolüsyonel sinir ağı için görselde iki göz, bir burun, bir ağız ve yüz ovali görmek yeterlidir. Bunların konumlarının bir önemi yoktur. Ayrıca doğru bir yüz görseli ters çevrilirse, içerik yine bir yüz olmasına rağmen konvolüsyonel sinir ağı bunu yüz olarak algılamaz.

Kapsül, bir nesnenin görselin belli bir bölgesinde bulunup bulunmadığını öğrenen nöronların oluşturduğu küçük bir gruptur ve bir verinin özelliklerinin skalar form yerine vektör formunda yakalanmasına olanak sağlar. Kapsüllerin çıktıları vektörlerdir. Vektörlerin uzunluğu nesnenin tahmini bulunma olasılığını belirler. Vektörün yönü ise nesnenin yerleşimiyle ilgili parametrelerdir. Nesnenin hafifçe yeri, boyutu değişse veya dönse, kapsülün çıktısı olan vektörün uzunluğu aynı kalır fakat yönü değişiklik gösterir. Bu durum yine de nesneyi tanımaya olanak sağlar. CapsNet, katmanlar arasında aktivasyon fonksiyonu olarak squash fonksiyonu kullanmaktadır. Bu fonksiyon vektörün uzunluğunu yani nesnenin bulunma olasılığını 0-1 arasında bir değere dönüştürür. Fakat vektörün yönünde bir değişiklik olmaz.

CapsNet, bir girdi katmanı, bir konvolüsyon katmanı ve bir öncül kapsül katmanından oluşmaktadır. Çalışmada CapsNet'i gerçeklemek için kullanılan kodlar Ek 1-5'te verilmiştir. Girdi katmanında 32x32 piksel boyutunda RGB görseller alınmış ve bu görsellere 9x9 boyutunda 256 adet filtre ile 1 atlamalı konvolüsyon uygulanmıştır. Ardından bu konvolüsyona ReLU aktivasyon işlemi uygulanmıştır. Bu işlemin sonucunda 24x24 boyutunda 256 boyutlu bir katman elde edilmiştir. Bir sonraki katman öncül kapsül katmanıdır. Bu katman, 8 boyutlu kapsüllerden meydana gelen 10x10 adet kapsül içeren 32 katmandan oluşmaktadır. Bu kapsüllerin görevi konvolüsyon katmanında öğrenilen basit özellikleri birleştirmektir. Son katmanda 2 adet şeffaflık kapsülü bulunmaktadır. Bunların her birinin boyutu 16'dır. Bir önceki katmandaki kapsüllerden, 16x8 boyutunda bir ağırlık matrisiyle çarpılarak elde edilmişlerdir. Bardakların şeffaflığı, bu kapsüllerin kayıp fonksiyonu tarafından hesaplanan kayıp değerlerine göre tahmin edilmektedir. Optimizasyon yöntemi olarak Adam optimizer (Kingma ve Ba 2014) kullanılmıştır.

4. BULGULAR

Bu bölümde, gerçekleştirilen dört farklı deneye ait bilgiler ve elde edilen sonuçlar sunulmuştur. Gerçekleştirilen deneylerin amacı nesnelerin şeffaflığını yakalamada CapsNet'in derin öğrenme yöntemlerine göre elde ettiği sonuçların analiz edilmesidir. Gerçekleştirilen deneyde aynı veri kümesi eğitim seti olarak kullanılarak LeNet, AlexNet, ResNet ve CapsNet ile eğitime tabi tutulmuştur. Çalışmada kullanılan veri kümesinin içerisinden rastgele %20'lik kısım test kümesi olarak ayrılmıştır. Rastgele seçilen 1200 adet görsel test verisi olarak kullanılmıştır. Veri kümesinden geri kalan görsellerin %20'lik kısmı doğrulama, %80'i ise eğitim sırasında kullanılmıştır. Böylece 960 veri ile doğrulama, 3839 veri ile eğitim yapılmıştır. Uygulamanın gerçekleştirilmesi için programlama dili olarak programlama dili olarak Python (Van Rossum ve Drake 2009) kullanılmıştır. Çalışmada kullanılan yöntemlerin uygulaması için TensorFlow (Abadi ve ark. 2016), Keras (Chollet 2015), OpenCV (Bradski 2000), NumPy (Walt ve ark. 2011) ve scikit-learn (Pedregosa ve ark. 2011) kütüphaneleri ile birlikte kullanılmıştır.

Bu çalışmada kullanılan yöntemler farklı boyutlarda veriler kabul ettiği için bazılarının girdi katmanlarında değişiklikler yapılmıştır. Bazıları için ise ağı besleyen verilerin boyutlarında değişikliğe gidilmiştir. Ayrıca, farklı problemleri çözmek için üretilmiş olan bu modeller, bu çalışmadaki sınıf sayısından farklı boyutlarda çıktı katmanlarına sahiptir. Bu yüzden, tüm modellerin çıktı katmanları iki sınıflı bir sınıflandırma yapacak şekilde değiştirilmiştir. Bazı modellerde bu değişiklikler ara katmanlarda da birtakım değişikliklere sebep olmuştur.

Elde edilen sonuçların değerlendirilmesi için 4 farklı metrik kullanılmıştır. Çalışmada kullanılan metrikler, Doğruluk, Kesinlik, Duyarlılık ve F-skoru'dur. Bu değerleri hesaplamak için scikit-learn kütüphanesinden faydalanılmıştır. Değerler her sınıftaki verilerin, veri kümesindeki bulunma miktarına göre ağırlıklı olarak hesaplanmıştır. Her bir sınıf için doğruluk değeri Denklem 4.1 ile hesaplanmaktadır. Burada Doğru Pozitif (DP) değeri, doğru bir şekilde pozitif olarak etiketlenen verileri, Doğru Negatif (DN) değeri, doğru bir şekilde negatif olarak etiketlenen verileri belirtmektedir. Toplam Pozitif

(TP) ve Toplam Negatif (TN) ise sırayla veri kümesi içindeki sınıfa ait pozitif ve negatif örnek sayısını temsil etmektedir.

$$A = \frac{DP + DN}{TP + TN} \tag{4.1}$$

Her bir sınıf için kesinlik değeri hesaplanırken Denklem 4.2 kullanılmaktadır. O sınıfa ait olarak etiketlenen örneklerden ne kadarının gerçekte o sınıfa ait olduğunu gösteren bir metriktir.

$$P = \frac{DP}{DP + YP} \tag{4.2}$$

Her bir sınıf için duyarlılık değeri hesaplanırken Denklem 4.3 kullanılmaktadır. Gerçekte o sınıfa ait örneklerden ne kadarının değerlendirme sırasında o sınıfa ait olarak etiketlendiğini gösteren bir metriktir.

$$R = \frac{DP}{DP + YN} \tag{4.3}$$

Her bir sınıf için F-skoru hesaplanırken Denklem 4.4 kullanılmaktadır. Kesinlik ve duyarlılık arasındaki dengeyi görmek için kullanılan bir metriktir. Yanlış Negatif (YN) ve yanlış pozitif (YP) değerleri bazı işlerde maliyetlere sebep olmaktadır. Veri kümesindeki sınıfların örnekleri içinde fazla doğru negatif olduğu durumlarda hataları daha net görmeye yaramaktadır.

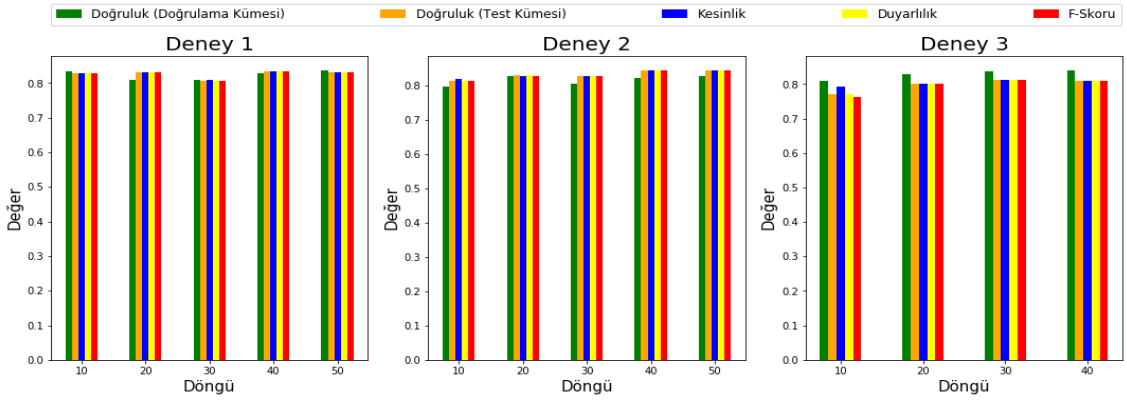
$$FS = 2 * \frac{P * R}{P + R} \tag{4.4}$$

Yapılan deneylerde LeNet modelinin üç farklı eğitim sırasında elde ettiği doğruluk (doğrulama ve test kümesi için), kesinlik, duyarlılık ve F-Skoru değerleri Çizelge 4.1'de verilmiştir. LeNet, 3 farklı deneyde 50 döngü boyunca eğitilmiştir. Devamında gelen döngülerde doğruluk değerlerinde iyileşme sağlanamadığı için eğitim durdurulmuştur.

Her 10 döngüde bir doğruluk değerleri kontrol edilmiştir. Doğrulama kümesindeki doğruluk sonuçlarına bakıldığında, her deneyde en yüksek doğrulama 50. döngünün sonunda alınmıştır. Bazı deneylerde sonra gelen döngüde doğruluk değeri düşmesine rağmen genel olarak döngü sayısı arttıkça doğruluk değeri yükselmiştir. 50 döngüye yaklaştıkça doğruluk değerindeki değişimler azalmıştır. Deneyler sırasında doğrulama kümesi üzerinde elde edilen en iyi doğruluk 0,8427'dir. Eğitim sonrası elde edilen en iyi test doğruluğu 0,8458'dir. Yapılan deneylerde alınan sonuçların grafik gösterimi Şekil 4.1'de verilmiştir.

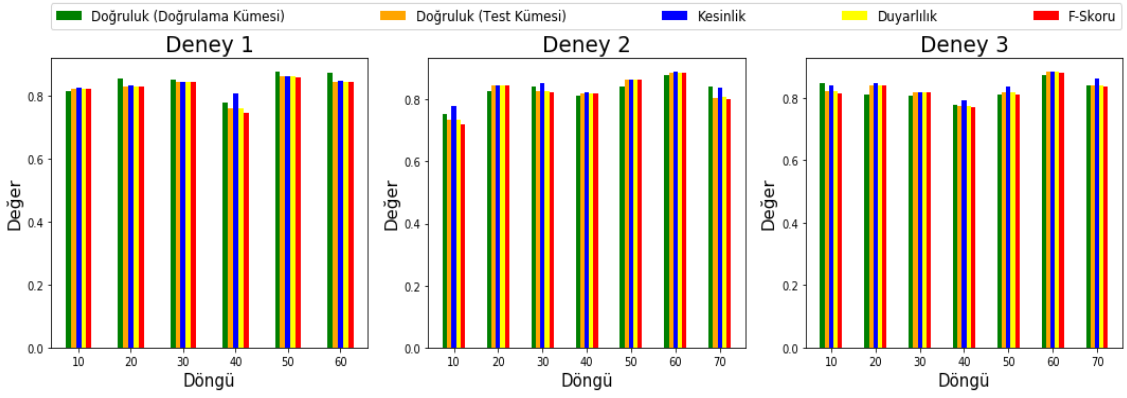
Çizelge 4.1. LeNet deney sonuçları

Deney	**Döngü Sayısı**	**Doğruluk**		**Kesinlik**	**Duyarlılık**	**F-Skoru**
		Doğrulama Kümesi	**Test Kümesi**			
1	10	0,8354	0,8283	0,8286	0,8283	0,8278
	20	0,8083	0,8308	0,8307	0,8308	0,8307
	30	0,8104	0,8058	0,8094	0,8058	0,8060
	40	0,8292	0,8342	0,8340	0,8341	0,8340
	50	0,8375	0,8325	0,8323	0,8325	0,8323
2	**10**	**0,7979**	**0,8142**	**0,8212**	**0,8141**	**0,8143**
	20	**0,8281**	**0,8300**	**0,8298**	**0,8300**	**0,8299**
	30	**0,8052**	**0,8283**	**0,8286**	**0,8283**	**0,8284**
	40	**0,8240**	**0,8458**	**0,8458**	**0,8458**	**0,8458**
	50	**0,8292**	**0,8442**	**0,8442**	**0,8441**	**0,8442**
3	10	0,8083	0,7708	0,7940	0,7708	0,7629
	20	0,8302	0,8025	0,8024	0,8025	0,8024
	30	0,8375	0,8117	0,8115	0,8116	0,8114
	40	0,8406	0,8100	0,8098	0,8100	0,8098



Şekil 4.1. LeNet deneylerinin grafik gösterimi

Yapılan deneylerde AlexNet modelinin üç farklı eğitim sırasında elde ettiği doğrulama, test, kesinlik, duyarlılık ve F skoru değerleri Çizelge 4.2'de verilmiştir. AlexNet, 3 farklı deneyde sırasıyla 60, 70 ve 70 döngü boyunca eğitilmiştir. Devamında gelen döngülerde doğruluk değerlerinde iyileşme sağlanamadığı için eğitim durdurulmuştur. Her 10 döngüde bir doğrulama kümesindeki doğruluk sonuçları kontrol edilmiştir. Doğrulama kümesinde doğruluk değerlerine bakıldığında, her 10 döngüde alınan doğruluk değerlerinin dalgalandığı görülmektedir. Döngü sayısı arttıkça doğruluk değerlerindeki iyileşmeler kadar azalmalar da görülmüştür. Deneyler sırasında doğrulama kümesi üzerinde elde edilen en iyi doğruluk 0,8781'dir. Eğitim sonrası elde edilen en iyi test doğruluğu 0,8858'dir. Yapılan deneylerde alınan sonuçların grafik gösterimi Şekil 4.2'de verilmiştir.



Şekil 4.2. AlexNet deneylerinin grafik gösterimi

Çizelge 4.2. AlexNet deney sonuçları

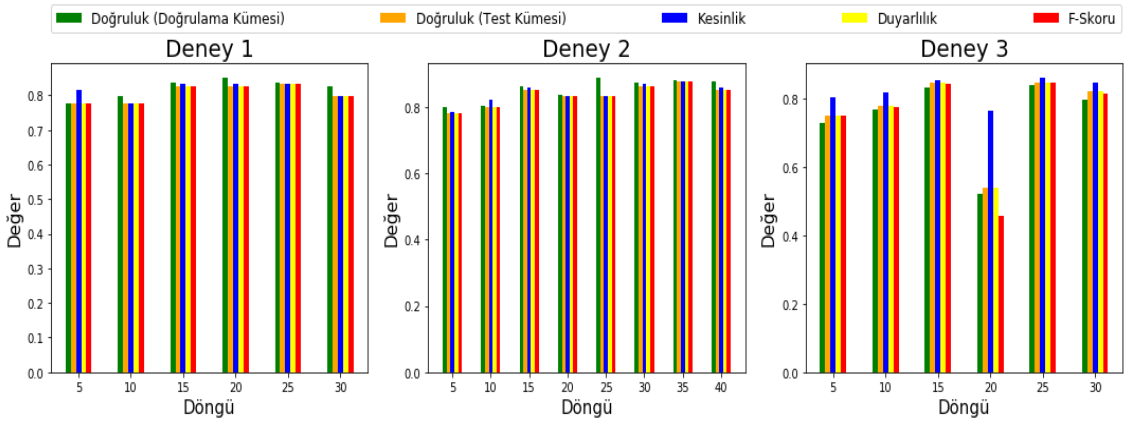
Deney	Döngü Sayısı	Doğruluk		Kesinlik	Duyarlılık	F-Skoru
		Doğrulama Kümesi	Test Kümesi			
1	10	0,8156	0,8250	0,8281	0,8250	0,8233
	20	0,8542	0,8292	0,8335	0,8291	0,8295
	30	0,8521	0,8467	0,8465	0,8466	0,8465
	40	0,7792	0,7617	0,8075	0,7616	0,7461
	50	0,8771	0,8617	0,8627	0,8616	0,8610
	60	0,8760	0,8450	0,8501	0,8450	0,8453
2	**10**	**0,7531**	**0,7350**	**0,7769**	**0,7350**	**0,7182**
	20	**0,8271**	**0,8458**	**0,8470**	**0,8458**	**0,8460**
	30	**0,8417**	**0,8267**	**0,8510**	**0,8266**	**0,8213**
	40	**0,8125**	**0,8200**	**0,8240**	**0,8200**	**0,8202**
	50	**0,8427**	**0,8642**	**0,8645**	**0,8641**	**0,8642**
	60	**0,8781**	**0,8858**	**0,8888**	**0,8858**	**0,8851**
	70	**0,8406**	**0,8038**	**0,8386**	**0,8083**	**0,8009**
3	10	0,8469	0,8208	0,8410	0,8208	0,8159
	20	0,8115	0,8417	0,8471	0,8416	0,8418
	30	0,8073	0,8175	0,8189	0,8175	0,8177
	40	0,7771	0,7725	0,7941	0,7725	0,7711
	50	0,8125	0,8167	0,8349	0,8166	0,8118
	60	0,8740	0,8825	0,8835	0,8825	0,8820
	70	0,8417	0,8400	0,8610	0,8400	0,8357

Yapılan deneylerde ResNet modelinin üç farklı eğitim sırasında elde ettiği doğrulama, test, kesinlik, duyarlılık ve F skoru değerleri Çizelge 4.3'te verilmiştir. ResNet, 3 farklı deneyde sırası ile 30, 40 ve 30 döngü boyunca eğitilmiştir. Devamında gelen döngülerde doğrulama ve test değerlerinde iyileşme sağlanamadığı için eğitim durdurulmuştur. Derin yapısı nedeniyle ResNet'te her döngü daha uzun sürmesine rağmen makul doğruluk sonuçlarına ulaşmak daha az sayıda döngüyle mümkün olmaktadır. Her 5 döngüde bir

doğrulama kümesinde doğruluk sonuçları kontrol edilmiştir. AlexNet'e benzer bir şekilde bu modelde de döngü sayısı arttıkça doğruluk değerlerinde dalgalanmalar görülmüştür. Deneyler sırasında elde edilen doğrulama kümesi üzerinde elde edilen iyi doğruluk 0,8865'tir. Eğitim sonrası elde edilen en iyi test doğruluğu 0,8758'dir. Yapılan deneylerde alınan sonuçların grafik gösterimi Şekil 4.3'te gösterilmiştir.

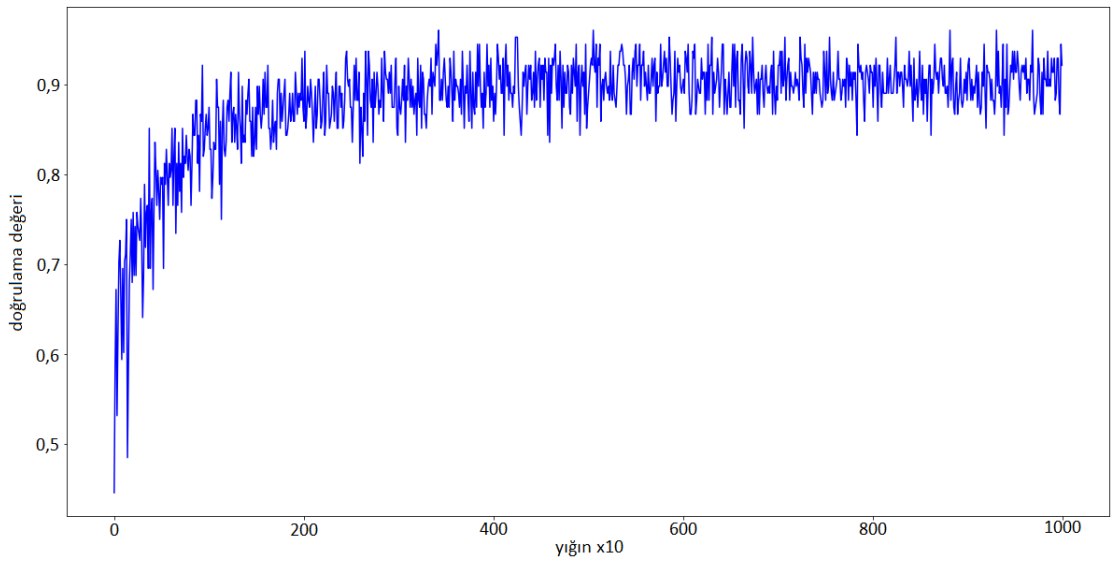
Çizelge 4.3. ResNet deney sonuçları

Deney	Döngü Sayısı	Doğruluk		Kesinlik	Duyarlılık	F-Skoru
		Doğrulama Kümesi	Test Kümesi			
1	5	0,7771	0,7783	0,8142	0,7783	0,7752
	10	0,7990	0,7758	0,7777	0,7758	0,7761
	15	0,8365	0,8267	0,8330	0,8266	0,8245
	20	0,8500	0,8275	0,8316	0,8275	0,8277
	25	0,8354	0,8325	0,8333	0,8325	0,8326
	30	0,8260	0,7983	0,7989	0,7983	0,7974
2	**5**	**0,7990**	**0,7792**	**0,7829**	**0,7791**	**0,7796**
	10	**0,8010**	**0,7992**	**0,8217**	**0,7991**	**0,7985**
	15	**0,8625**	**0,8500**	**0,8567**	**0,8500**	**0,8503**
	20	**0,8354**	**0,8308**	**0,8315**	**0,8308**	**0,8310**
	25	**0,8865**	**0,8308**	**0,8315**	**0,8308**	**0,8310**
	30	**0,8729**	**0,8625**	**0,8705**	**0,8625**	**0,8627**
	35	**0,8812**	**0,8758**	**0,8774**	**0,8758**	**0,8760**
	40	**0,8760**	**0,8508**	**0,8561**	**0,8508**	**0,8511**
3	5	0,7292	0,7517	0,8027	0,7516	0,7483
	10	0,7688	0,7783	0,8176	0,7783	0,7768
	15	0,8323	0,8467	0,8545	0,8466	0,8437
	20	0,5229	0,5383	0,7655	0,5383	0,4574
	25	0,8385	0,8458	0,8601	0,8458	0,8462
	30	0,7979	0,8217	0,8467	0,8216	0,8142



Şekil 4.3. ResNet deneylerinin grafik gösterimi

CapsNet modeli, hafıza problemlerini aşmak adına eğitim verilerinin tümünü döngülerde kullanmak yerine, eğitim verilerinin arasından rastgele seçilen görsellerin oluşturduğu yığınlarla (batch) beslenerek eğitilmiştir. Yapılan deneylerde CapsNet modelinin eğitimi 128 verilik yığınlarla 10000 yığın boyunca sürmüştür. Devamında gelen yığınlarda doğrulama değerinde iyileşme sağlanamadığı için eğitim durdurulmuştur. Eğitim sırasında her yığında elde ettiği doğrulama değerleri Şekil 4.4'te verilmiştir. Deneyler sırasında elde edilen en iyi test doğruluğu 0,9158'dir. CapsNet ile yapılan sınıflandırmanın karmaşıklık matrisi Çizelge 4.4'te verilmiştir. Elde edilen kesinlik, duyarlılık ve F skoru değerleri sırası ile 0,9161, 0,9158 ve 0,9156'dır.



Şekil 4.4. CapsNet eğitiminde doğrulama değerleri

Çizelge 4.4. CapsNet karmaşıklık matrisi

	Şeffaf değil olarak sınıflandırılan	**Şeffaf olarak sınıflandırılan**
Şeffaf olmayan nesne	627	38
Şeffaf nesne	63	472

Veri kümesinde şeffaf sınıfında bira ve su bardaklarının sayısı daha fazla iken şeffaf olmayanlar sınıfında kupa türündeki örneklerin sayısı fazladır. Farklı sınıflardaki bu nesneler sadece saydamlığı değil şekilleri açısından da farklara sahiptir. CapsNet'in başarısının farklı sınıflardaki nesnelerin şekil olarak birbirinden farklı olmasından değil, nesnelerin transparanlığını algılamasından kaynaklandığını test etmek için iki sınıftan birbirine benzer nesneler seçilerek testler yapılmıştır. Yapılan testlerde birbirine benzeyen şeffaf olan ve olmayan nesneleri doğru sınıflandırdığı görülmüştür. Bu nesnelerin örnekleri Şekil 4.5'te verilmiştir.

(A) (B)

Şekil 4.5. Doğru sınıflandırılan benzer görünümlü nesneler **A)** Şeffaf olmayan **B)** Şeffaf

Şeffaf olmayan örnekler çoğu zaman arka planlarından renk olarak ayrışmaktadır. Kurulan sistemin, şeffaf olan ve olmayan nesneleri, şeffaflıklarına göre değil, arka planlarının rengiyle aynı olup olmamasına göre sınıflandırma ihtimali bulunmaktadır. Bu ayrımın yapılabilmesi için şeffaf nesnelerden içi renkli sıvı dolu olanlar seçilip yapılan sınıflandırmanın sonucuna bakılmıştır. Ayrıca arka planında farklı renk geçişleri olan

nesnelerde bu geçişlerin, nesnenin üzerindeki desenler olarak algılanma ihtimali bulunmaktadır. Böyle bir arka plana sahip şeffaf nesne seçilerek sınıflandırma sonucu incelenmiştir. Her iki testte de doğru sınıflandırmalar yapılmıştır. Bu sınıflandırmalarda kullanılan verilerin örnekleri Şekil 4.6'da verilmiştir.

(A) (B)

Şekil 4.6. Doğru sınıflandırılan şeffaf nesneler **A)** Renkli bir sıvı ile dolu **B)** Renkli bir arka plana sahip

CapsNet'in başarısız olduğu veriler araştırıldığında yarı saydam sayılabilecek nesneler öne çıkmıştır. Bu nesneler insanlar tarafından sınıflandırılırken de farklı yorumlanabilecek nesnelerdir. Bu verilere ait örnekler Şekil 4.7'de verilmiştir.

Şekil 4.7. Yarı saydam nesne örneği

5. TARTIŞMA ve SONUÇ

Gerçekleştirilen çalışma bardakların şeffaflığının farklı metotlar yardımıyla tespit edilmesi üzerinedir. Bunu gerçekleştirebilmek için yeni geliştirilen CapsNet ile mevcutta uzun yıllardır kullanılan derin öğrenme yöntemlerini (LeNet, AlexNet, ResNet) karşılaştırabilmek için bir deney ortamı oluşturulmuştur. Çalışmada kullanılmak üzere elde edilen veriler üzerinde, kullanılacak derin öğrenme modelleriyle uyumlu olmaları için çeşitli ön-işlemler yapıldıktan sonra 4 farklı yöntem için çalışmalar gerçekleştirilmiştir.

Yapılan deneyler sonucunda en düşük doğruluk değerlerini sağlayan yapı LeNet olmuştur. Ortaya çıkış sebebi el yazısı rakamları tanımak olan ve 10 sınıf için tahmin yapan LeNet'in diğer ağlara göre basit olan yapısı, şeffaflığın tespiti konusunda diğer ağlara göre başarısız olmasının nedeni olarak söylenebilir.

Diğer iki derin öğrenme modeli olan AlexNet ve ResNet birbirine yakın sonuçlar elde etmiştir. Test doğruluğunda AlexNet bir adım öne çıkmıştır. ImageNet yarışmasının 1000 sınıflık problemlerini sınıflandırmak için üretilen bu iki modelde AlexNet, şeffaflık tespiti konusunda ResNet'ten daha iyi sonuçlar vermiştir. Bir nesnenin şeffaf olup olmadığını algılama konusunda ResNet'in çok sayıda katman içeren derin yapısı AlexNet'e göre başarısız kalmıştır. Ağın derinleşmesinin bir nesnenin şeffaflığını algılama konusunda büyük bir avantajının olmadığı görülmüştür.

Nesne tanıma problemlerine yeni ve etkili bir çözüm getiren CapsNet, bu çalışmada diğer derin öğrenme yöntemlerine göre daha iyi sonuçlar elde etmiştir. Klasik derin öğrenme yöntemlerinde bulunan katmanlar iki boyutludur. CapsNet'te bu durum üç boyuta yükselmiştir. Bu durum, aynı nesne ters çevrildiğinde (baş aşağı) diğer derin öğrenme yöntemleri sınıflandırmada başarısız olurken, CapsNet'in doğru sınıflandırmasını sağlamaktadır. Aynı zamanda bir nesnenin parçaları olmaması gereken konumlarda iken diğer derin öğrenme yöntemleri mevcut örneği, o nesne olarak tanımaktadır. CapsNet'te bu durumun önüne geçilmiş ve nesnenin parçalarının da doğru konumda olup olmadıklarına göre sınıflandırma sağlanmıştır. Kapsüllerin bu üç boyutlu yapısının, farklı

nesneleri sınıflandırmanın yanında bir nesnenin şeffaf olup olmadığını algılama konusunda da başarıya katkı sağladığı görülmüştür.

CapsNet her ne kadar bu çalışmada çok başarılı bir sonuç elde etse de mevcut durumda sınıflandırmada başarısız olduğu nesneler incelendiğinde bunların yarı saydam nesneler olduğu görülmüştür. Gelecek çalışmalarda bu tür nesneler için yeni yarı saydam nesne sınıfı eklenecek ve doğruluğu arttırmaya yönelik değişiklikler yapılacaktır.

Çalışmada, derin öğrenme modellerinin girdisi olarak kullanılan eğitim görselleri gerek hafıza problemleri gerek modellerin birbirine uyumlu olması açısından küçük boyutlarda kullanılmıştır. Hafıza problemlerinin aşılacağı bir donanımda, CapsNet'in yapısı daha büyük boyutlu girdi görselleri kabul edecek şekilde değiştirilecektir. Bu yeni model eğitim verilerinin daha büyük halleriyle eğitilecek ve girdi görsellerinin ayrıntısının sınıflandırma başarısı üzerindeki etkisi test edilecektir.

Nesnelerin şeffaflığını ön plana çıkarabilecek filtreler ve görüntü işleme yöntemleri araştırılacaktır. Bu yöntemler ile ön işlemden geçirilen eğitim verileri şeffaflığı vurgulanmış hale getirilecektir. CapsNet modeli bu yeni verilerle beslenerek eğitilecek ve bu ön işlemlerin sınıflandırma başarısı üzerindeki etkisi gözlenecektir. Daha başarılı sonuçlar elde edilirse bu filtreler CapsNet'in giriş katmanının öncesine eklenerek yeni bir yapı oluşturulacaktır.

Bardak dışında, şeffaf ve şeffaf olmayan şişe, vazo gibi farklı nesneler için eğitim verileri toplanacaktır. Bu veriler hem nesnelerin türüne hem de şeffaflığına göre sınıflara ayrılarak eğitim kümesi genişletilecektir. CapsNet'in çıktı katmanı mevcut veri kümesindeki sınıflara göre düzenlenecektir. Böylece ağın, nesneleri hem türüne hem de şeffaflığına göre sınıflandırdığı bir yapı oluşturulmaya çalışılacaktır.

KAYNAKLAR

Abadi, M., Barham, P., Chen, J., Chen, Z., Davis, A., Dean, J., Devin, M., Ghemawat, S., Irving, G., Isard, M., Kudlur, M., Levenberg, J., Monga, R., Moore, S., Murray, D.G., Steiner, B., Tucker, P., Vasudevan, V., Warden, P., Wicke, M., Yu, Y., Zheng, X. 2016. Tensorflow: A system for large-scale machine learning. The 12th USENIX Symposium on Operating Systems Design and Implementation, 2-4 Kasım 2016, Savannah, ABD.

Afshar, P., Mohammadi, A., Plataniotis, K.N. 2018. Brain tumor type classification via capsule networks. The 25th IEEE International Conference on Image Processing, 7-10 Ekim 2018, Atina, Yunanistan.

Akbay, Y.E. 2017. İdealar teorisi bağlamında Planton'da akıl ilkelerinin analizi. *Süleyman Demirel Üniversitesi Sosyal Bilimler Enstitüsü Dergisi*, 28(3): 133-155.

Akgündüz, G.Ö. 2019. Platon'un Timaios diyalogunda akıl-zorunluluk ilişkisi. *Beytulhikme: An International Journal of Philosophy*, 9(3): 625-650.

Alpaydın, E. 2010. Introduction to machine learning. MIT Press, Massachusetts, 537 s.

Anonim, 2006. Al-Jazari - A Musical Toy. Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Al-Jazari_-_A_Musical_Toy.jpg- (Erişim tarihi: 25.05.2020)

Anonim, 2010. Didrachm Phaistos obverse. Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Didrachm_Phaistos_obverse_CdM.jpg- (Erişim tarihi: 25.05.2020)

Anonim, 2016. Enki (Ea). Wikimedia Commons, https://commons.wikimedia.org/wiki/File:Enki(Ea).jpg?uselang=tr- (Erişim tarihi: 10.05.2020).

Anonim, 2019a. Class CategoricalCrossentropy. TensorFlow Core Documentation, https://www.tensorflow.org/versions/r1.15/api_docs/\\python/tf/keras/losses/CategoricalCrossentropy- (Erişim tarihi: 07.06.2019)

Anonim, 2019b. Class BinaryCrossentropy. Tensorflow Core Documentation, https://www.tensorflow.org/versions/r1.15/api_docs/\\python/tf/keras/losses/BinaryCrossentropy- (Erişim tarihi: 07.06.2019)

Anonim, 2020a. Güncel Türkçe Sözlük. Türk Dil Kurumu sözlükleri, https://sozluk.gov.tr/- (Erişim tarihi: 09.05.2020)

Anonim, 2020b. İskenderiyeli Heron. Vikipedi, https://tr.wikipedia.org/wiki/%C4%B0skenderiyeli_Heron- (Erişim tarihi: 24.05.2020)

Anonim, 2020c. Ismail al-Jazari. Wikipedia, https://en.wikipedia.org/wiki/Ismail_al-Jazari- (Erişim tarihi: 25.05.2020)

Anonim, 2020d. Choosing the right estimator — scikit-learn 0.23.1 documentation. Scikit-learn. https://scikit-learn.org/stable/tutorial/machine_learning_map/index.html- (Erişim tarihi: 03.06.2020)

Bae, J., Kim, D. 2018. End-to-end speech command recognition with capsule network. Interspeech, 2-6 Eylül 2018, Haydarabad, Hindistan.

Basheer, I.A., Hajmeer, M. 2000. Artificial neural networks: fundamentals, computing, design, and application. *Journal of microbiological methods:* 43(1): 3-31.

Bayık, F. 2019. Aristoteles ve Descartes bağlamında akıl ve zekâ kavramlarının farkları. *Kaygı. Uludağ Üniversitesi Fen-Edebiyat Fakültesi Felsefe Dergisi*, 18(1): 172-187.

Bertok, K. 2016. OpenCV - Using SVM and HOG for person detection, Stack Overflow, https://stackoverflow.com/questions/34985196/opencv-using-svm-and-hog-for-person-detection- (Erişim tarihi: 21.06.2020)

Bilgin, M., Mutludoğan, K. 2019. American sign language character recognition with capsule networks. 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies, 11-13 Ekim 2019, Ankara, Türkiye.

Birand, K. 1958. İlk çağ felsefesi tarihi. Ankara Üniversitesi İlahiyat Fakültesi Yayınları, Ankara, 135 s.

Bradski, G. 2000. The opencv library. *Dr Dobb's Journal of Software Tools*, 25: 120-125.

Bradski, G., Kaehler, A. 2008. Learning opencv. O'Reilly Media, Inc., ABD, 555 s.

Buchanan, B.G. 2005. A (very) brief history of artificial intelligence. *Ai Magazine*, 26(4): 53-60.

Chollet, F. 2015. Keras. Keras: the Python deep learning API, https://keras.io- (Erişim tarihi: 15.05.2019).

Cianciolo, A.T., Sternberg, R.J. 2004. Intelligence: a brief history. Blackwell Publishing, Avustralya, 170 s.

Çığ, M.İ. 2018. Uygarlığın kökeni Sümerliler – 1, Tarihte ilk edebi eserlerden seçmeler. Kaynak Yayınları, İstanbul, 292 s.

Dalal, N., Triggs, B. 2005. Histograms of oriented gradients for human detection. IEEE Computer Society Conference on Computer Vision and Pattern Recognition, 20-25 Haziran 2005, San Diego, ABD.

Deng, J., Dong, W., Socher, R., Li, L., Li, K., Fei-Fei, L. 2009. Imagenet: A large-scale hierarchical image database. IEEE Conference on Computer Vision and Pattern Recognition, 20-25 Haziran 2009, Miami, ABD.

Deshpande, J. 2015. Pierre Jaquet-Droz, Marvel Maker: The Man Behind Today's Jaquet-Droz Watch Brand. WatchTime, https://www.watchtime.com/featured/pierre-jaquet-droz-marvel-maker-the-man-behind-todays-jaquet-droz-watch-brand/- (Erişim tarihi: 25.05.2020)

Forsyth, D.A., Ponce, J. 2012. Computer vision: A modern approach. Pearson Education, Inc., ABD, 761 s.

Gatys, L.A., Ecker, A.S., Bethge, M. 2016. Image style transfer using convolutional neural networks. IEEE Conference on Computer Vision and Pattern Recognition, 27-30 Haziran 2016, Las Vegas, ABD.

Goodfellow, I., Bengio, Y., Courville, A. 2016. Deep learning. MIT Press, ABD, 781 s.

Gupte, S., Masoud, O., Martin, R.F.K., Papanikolopoulos, N.P. 2002. Detection and classification of vehicles. *IEEE Transactions on Intelligent Transportation Systems,* 3(1): 37-47.

Gürel, E., Tat, M. 2010. Çoklu zekâ kuramı: Tekli zekâ anlayışından çoklu zekâ yaklaşımına. *Journal of International Social Research*, 3(11): 336-356.

Hart, M. 2020. Tesla's autopilot 'vision' looks like terminator hud. Nerdist, https://nerdist.com/article/teslas-autopilot-vision-terminator-hud/- (Erişim tarihi: 09.06.2020)

He, K., Zhang, X., Ren, S., Sun, J. 2016. Deep residual learning for image recognition. IEEE Conference on Computer Vision and Pattern Recognition, 27-30 Haziran 2016, Las Vegas, ABD.

Hebb, D.O. 1949. The organization of behavior: A neuropsychological theory. John Wiley & Sons Inc., New York, 335 s.

Kaelbling, L.P., Littman, M.L., Moore, A.W. 1996. Reinforcement learning: A survey. *Journal of Artificial Intelligence Research,* 4: 237-285.

Kelleci, A. 2011. Farabi'de akıl kavramı. *Yüksek Lisans Tezi*, Maltepe Üniversitesi Sosyal Bilimler Enstitüsü, Felsefe Anabilim Dalı, İstanbul.

Khaing, M.P., Masayuki, M. 2018. Transparent object detection using convolutional neural network. International Conference on Big Data Analysis and Deep Learning Applications, 14-15 Mayıs 2018, Miyazaki, Japonya.

Kılıç, C. 2014. John Locke: Bilginin kaynağı ve ideler sorunu. *Ekev Akademi Dergisi*, 58(58): 455-468.

Kılıç, Y. 2015. Hobbes, Locke ve Rousseau'da "Doğa Durumu" düşüncesi. *Temaşa Erciyes Üniversitesi Felsefe Bölümü Dergisi*, 2: 97-117.

Kırlı, Ö. 2013. John Locke ve David Hume'un epistemolojisi ve beşeri olanın izahı. *Uluslararası Yönetim İktisat ve İşletme Dergisi*, 9(20): 99-114.

Kingma, D.P., Ba, J. 2014. Adam: A method for stochastic optimization. *arXiv preprint*, arXiv:1412.6980.

Klank, U., Carton, D., Beetz, M. 2011. Transparent object detection and reconstruction on a mobile platform. IEEE International Conference on Robotics and Automation, 9-13 Mayıs 2011, Şangay, Çin.

Kompella, V.R., Sturm, P. 2011. Detection and avoidance of semi-transparent obstacles using a collective-reward based approach. IEEE International Conference on Robotics and Automation, 9-13 Mayıs 2011, Şangay, Çin.

Krizhevsky, A., Sutskever, I., Hinton, G.E. 2012. Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 3-8 Aralık 2012, Lake Tahoe, ABD.

LeCun, Y., Bottou, L., Bengio, Y., Haffner, P. 1998. Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, 86(11): 2278-2324.

LeCun, Y., Cortes, C. 2010. MNIST handwritten digit database. The MNIST Database of Handwritten Digits, http://yann.lecun.com/exdb/mnist/- (Erişim tarihi: 17.05.2019)

LeCun, Y., Bengio, Y., Hinton, G. 2015. Deep learning. *Nature*, 521(7553): 436-444.

Li, Y., Qian, M., Liu, P., Cai, Q., Li, X., Guo, J., Yan, H., Yu, F., Yuan, K., Yu, J. 2019. The recognition of rice images by UAV based on capsule network. *Cluster Computing*, 22(4): 9515-9524.

Litjens, G., Kooi, T., Bejnordi, B.E., Setio, A.A.A., Ciompi, F., Ghafoorian, M., Van Der Laak, J.A., Van Ginneken, B., Sanchez, C.I. 2017. A survey on deep learning in medical image analysis. *Medical Image Analysis*, 42: 60-88.

Mayor, A. 2018. Gods and robots: Myths, machines, and ancient dreams of technology. Princeton University Press, ABD, 275 s.

McCorduck, P., Minsky, M., Selfridge, O.G., Simon, H.A. 1977. History of artificial intelligence. Fifth International Joint Conference on Artificial Intelligence, 22-25 Ağustos 1977, Massachusetts, ABD.

McCorduck, P. 2004. Machines who think: A personal inquiry into the history and prospects of artificial intelligence. A K Peters, Ltd., Kanada, 553 s.

McCulloch, W.S., Pitts, W. 1943. A logical calculus of the ideas immanent in nervous activity. *The bulletin of mathematical biophysics,* 5(4): 115-133.

Mobiny, A., Van Nguyen, H. 2018. Fast capsnet for lung cancer screening. International Conference on Medical Image Computing and Computer-Assisted Intervention, 16-20 Eylül 2018, Granada, İspanya.

Nair, V., Hinton, G.E. 2010. Rectified linear units improve restricted boltzmann machines. The 27th International Conference on Machine Learning, 21-24 Haziran 2010, Hayfa, İsrail.

Nasr, G.E., Badr, E.A., Joun, C. 2002. Cross entropy error function in neural networks: Forecasting gasoline demand. The 15th International Florida Artificial Intelligence Research Society Conference, 14-16 Mayıs 2002, Florida, ABD.

Öktem, Ü. 2004. David Hume ve Immanuel Kant'ın kesin bilgi anlayışı. *Ankara Üniversitesi Dil ve Tarih-Coğrafya Fakültesi Dergisi*, 44(2): 29-55.

Pedregosa, F., Varoquaux, G., Gramfort, A., Michel, V., Thirion, B., Grisel, O., Blondel, M., Prettenhofer, P., Weiss, R., Dubourg, V., Vanderplas, J., Passos, A., Cournapeau, D., Brucher, M., Perrot, M., Duchesnay, E. 2011. Scikit-learn: Machine learning in Python. *Journal of Machine Learning Research*, 12: 2825-2830.

Ray, S. 2018. History of AI. Towards Data Science, https://towardsdatascience.com/history-of-ai-484a86fc16ef- (Erişim tarihi: 30.05.2020)

Roberts, S. 2018. From Homer to HAL: 3,000 years of AI narratives. *Research Horizons*, 35: 28-29.

Rosenberg, R.S. 1992. The social impact of computers. Academic Press, Inc., ABD, 375 s.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. 2015. Imagenet large scale visual recognition challenge. *International Journal of Computer Vision*, 115(3): 211-252.

Russell, S., Norvig, P. 2002. Artificial intelligence: a modern approach. Pearson Education, ABD, 1132 s.

Sabour, S., Frosst, N., Hinton, G.E. 2017. Dynamic routing between capsules. Advances in Neural Information Processing Systems, 4-9 Aralık 2017, Long Beach, ABD.

Singh, N. 2018. Artificial intelligence and it's sub-fields. Medium, https://medium.com/@neha49712/artificial-intelligence-and-its-sub-fields-a5a63d8263e8- (Erişim tarihi: 30.05.2020)

Smith, G.W., Leymarie, F.F. 2017. The machine as artist: An introduction. *Arts*, 6(2): 5.

Sternberg, R.J. 1990. Metaphors of mind: Conceptions of the nature of intelligence. Cambridge University Press, Kanada, 344 s.

Szeliski, R. 2011. Computer vision: Algorithms and applications. Springer, ABD, 812 s.

Turing, A.M. 1950. Computing machinery and intelligence. *Mind,* 59(236): 433-460.

Turovsky, B. 2016. Found in translation: More accurate, fluent sentences in Google Translate. Google Blog, https://blog.google/products/translate/found-translation-more-accurate-fluent-sentences-google-translate/- (Erişim tarihi: 08.06.2020)

Uysal, E. 2004. Kindî ve Fârâbî'de akıl ve nefs kavramlarının ahlâkî içeriği. *Uludağ Üniversitesi İlahiyat Fakültesi Dergisi*, 13(2): 141-156.

Van Rossum, G., Drake, F.L. 2009. Python 3 Reference Manual. CreateSpace Independent Publishing Platform, Scotts Valley, 242 s.

Walt, S.v.d., Colbert, S.C., Varoquaux, G. 2011. The NumPy array: a structure for efficient numerical computation. *Computing in Science & Engineering,* 13(2): 22-30.

Wang, Y., Sun, A., Han, J., Liu, Y., Zhu, X. 2018. Sentiment analysis by capsules. The World Wide Web Conference, 23-27 Nisan 2018, Lyon, Fransa.

Yao, X. 1999. Evolving artificial neural networks. *Proceedings of the IEEE,* 87(9): 1423-1447.

EKLER

EK 1 CapsNet Modelini Oluşturan Parçaların Python Dilindeki Kodları

EK 2 CapsNet Ana Modelinin Python Dilinde Oluşturulması

EK 3 CapsNet Modelinde Kullanılan “model_base.py” Dosyası

EK 4 CapsNet Modelinde Kullanılan “utils.py” Dosyası

EK 5 CapsNet Modelinde Kullanılan “logger.py” Dosyası

EK 1 CapsNet Modelini Oluşturan Parçaların Python Dilindeki Kodları

Bu çalışmada CapsNet modelinin gerçeklenmesi için https://github.com/thibo73800/capsnet-traffic-sign-classifier adresindeki Github deposundan faydalanılmıştır.

```
import numpy as np
import tensorflow as tf


def conv_caps_layer(input_layer, capsules_size, nb_filters, kernel,
stride=2):
    """
        Capsule layer for the convolutional inputs
        **input:
            *input_layer: (Tensor)
            *capsule_numbers: (Integer) the number of capsule in th
is layer.
            *kernel_size: (Integer) Size of the kernel for each fil
ter.
            *stride: (Integer) 2 by default
    """
    # "In convolutional capsule layers each unit in a capsule is a
convolutional unit.
    # Therefore, each capsule will output a grid of vectors rather
than a single vector output."
    capsules = tf.contrib.layers.conv2d(
        input_layer, nb_filters * capsules_size, kernel, stride, pa
dding="VALID")
    # conv shape: [?, kernel, kernel, nb_filters]
    shape = capsules.get_shape().as_list()
    capsules = tf.reshape(capsules, shape=(-1, np.prod(shape[1:3])
* nb_filters, capsules_size, 1))
    # capsules shape: [?, nb_capsules, capsule_size, 1]
    return squash(capsules)

def routing(u_hat, b_ij, nb_capsules, nb_capsules_p, iterations=4):
    """
        Routing algorithm

        **input:
            *u_hat: Dot product (weights between previous capsule a
nd current capsule)
            *b_ij: the log prior probabilities that capsule i shoul
d be coupled to capsule j
            *nb_capsules_p: Number of capsule in the previous layer
            *nb_capsules: Number of capsule in this layer
    """
    # Start the routing algorithm
    for it in range(iterations):
        with tf.variable_scope('routing_' + str(it)):
```

```
            # Line 4 of algo
            # probabilities that capsule i should be coupled to cap
sule j.
            # c_ij:  [nb_capsules_p, nb_capsules, 1, 1]
            c_ij = tf.nn.softmax(b_ij, dim=2)

            # Line 5 of algo
            # c_ij:  [      nb_capsules_p, nb_capsules, 1,
1]
            # u_hat: [?,    nb_capsules_p, nb_capsules, len_v_j,
1]
            s_j = tf.multiply(c_ij, u_hat)
            # s_j: [?, nb_capsules_p, nb_capsules, len_v_j, 1]
            s_j = tf.reduce_sum(s_j, axis=1, keep_dims=True)
            # s_j: [?, 1, nb_capsules, len_v_j, 1)

            # line 6:
            # squash using Eq.1,
            v_j = squash(s_j)
            # v_j: [1, 1, nb_capsules, len_v_j, 1)

            # line 7:
            # Frist reshape & tile v_j
            # [? ,  1,              nb_capsules,    len_v_j, 1] ->
            # [?,   nb_capsules_p,  nb_capsules,    len_v_j, 1]
            v_j_tiled = tf.tile(v_j, [1, nb_capsules_p, 1, 1, 1])
            # u_hat:    [?,             nb_capsules_p, nb_capsules,
len_v_j, 1]
            # v_j_tiled [1,             nb_capsules_p, nb_capsules,
len_v_j, 1]
            u_dot_v = tf.matmul(u_hat, v_j_tiled, transpose_a=True)
            # u_produce_v: [?, nb_capsules_p, nb_capsules, 1, 1]
            b_ij += tf.reduce_sum(u_dot_v, axis=0, keep_dims=True)
            #b_ih: [1, nb_capsules_p, nb_capsules, 1, 1]

    return tf.squeeze(v_j, axis=1)

def fully_connected_caps_layer(input_layer, capsules_size, nb_capsu
les, iterations=4):
    """
        Second layer receiving inputs from all capsules of the laye
r below
            **input:
                *input_layer: (Tensor)
                *capsules_size: (Integer) Size of each capsule
                *nb_capsules: (Integer) Number of capsule
                *iterations: (Integer) Number of iteration for the
routing algorithm

            i refer to the layer below.
            j refer to the layer above (the current layer).
    """
    shape = input_layer.get_shape().as_list()
    # Get the size of each capsule in the previous layer and the cu
rrent layer.
    len_u_i = np.prod(shape[2])
```

```
    len_v_j = capsules_size
    # Get the number of capsule in the layer bellow.
    nb_capsules_p = np.prod(shape[1])

    # w_ij: Used to compute u_hat by multiplying the output ui of a
capsule in the layer below
    # with this matrix
    # [nb_capsules_p, nb_capsules, len_v_j, len_u_i]
    _init = tf.random_normal_initializer(stddev=0.01, seed=0)
    _shape = (nb_capsules_p, nb_capsules, len_v_j, len_u_i)
    w_ij = tf.get_variable('weight', shape=_shape, dtype=tf.float32
, initializer=_init)

    # Adding one dimension to the input [batch_size, nb_capsules_p,
length(u_i), 1] ->
    #                                   [batch_size, nb_capsules_p,
1, length(u_i), 1]
    # To allow the next dot product
    input_layer = tf.reshape(input_layer, shape=(-1, nb_capsules_p,
1, len_u_i, 1))
    input_layer = tf.tile(input_layer, [1, 1, nb_capsules, 1, 1])

    # Eq.2, calc u_hat
    # Prediction uj|i made by capsule i
    # w_ij:  [              nb_capsules_p, nb_capsules, len_v_j,  l
en_u_i, ]
    # input: [batch_size,   nb_capsules_p, nb_capsules, len_ui,   1
]
    # u_hat: [batch_size,   nb_capsules_p, nb_capsules, len_v_j, 1]
    # Each capsule of the previous layer capsule layer is associate
d to a capsule of this layer
    u_hat = tf.einsum('abdc,iabcf->iabdf', w_ij, input_layer)

    # bij are the log prior probabilities that capsule i should be
coupled to capsule j
    # [nb_capsules_p, nb_capsules, 1, 1]
    b_ij = tf.zeros(shape=[nb_capsules_p, nb_capsules, 1, 1], dtype
=np.float32)

    return routing(u_hat, b_ij, nb_capsules, nb_capsules_p, iterati
ons=iterations)

def squash(vector):
    """
        Squashing function corresponding to Eq. 1
        **input: **
            *vector
    """
    vector += 0.00001 # Workaround for the squashing function ...
    vec_squared_norm = tf.reduce_sum(tf.square(vector), -2, keep_di
ms=True)
    scalar_factor = vec_squared_norm / (1 + vec_squared_norm) / tf.
sqrt(vec_squared_norm)
    vec_squashed = scalar_factor * vector  # element-wise
    return(vec_squashed)
```

EK 2 CapsNet Ana Modelinin Python Dilinde Oluşturulması

Bu çalışmada CapsNet modelinin gerçeklenmesi için https://github.com/thibo73800/capsnet-traffic-sign-classifier adresindeki Github deposundan faydalanılmıştır.

```
#!/usr/bin/python3
# -*- coding: utf-8 -*-

import numpy as np
from model_base import ModelBase
import tensorflow as tf

class ModelTrafficSign(ModelBase):
    """
        ModelTrafficSign.
        This class is used to create the conv graph using:
            Dynamic Routing Between Capsules
    """

    # Numbers of label to predict
    NB_LABELS = 2

    def __init__(self, model_name, output_folder):
        """
            **input:
                *model_name: (Integer) Name of this model
                *output_folder: Output folder to saved data (tensor
board, checkpoints)
        """
        ModelBase.__init__(self, model_name, output_folder=output_f
older)

    def _build_inputs(self):
        """
            Build tensorflow inputs
            (Placeholder)
            **return: **
                *tf_images: Images Placeholder
                *tf_labels: Labels Placeholder
        """
        # Images 32*32*3
        tf_images = tf.placeholder(tf.float32, [None, 32, 32, 3], n
ame='images')
        # Labels: [0, 1, 6, 20, ...]
        tf_labels = tf.placeholder(tf.int64, [None], name='labels')
        return tf_images, tf_labels

    def _build_main_network(self, images, conv_2_dropout):
        """
```

```
            This method is used to create the two convolutions and
the CapsNet on the top
            **input:
                *images: Image PLaceholder
                *conv_2_dropout: Dropout value placeholder
            **return: **
                *Caps1: Output of first Capsule layer
                *Caps2: Output of second Capsule layer
        """
        # First BLock:
        # Layer 1: Convolution.
        shape = (self.h.conv_1_size, self.h.conv_1_size, 3, self.h.
conv_1_nb)
        conv1 = self._create_conv(self.tf_images, shape, relu=True,
max_pooling=False, padding='VALID')
        # Layer 2: Convolution.
        #shape = (self.h.conv_2_size, self.h.conv_2_size, self.h.co
nv_1_nb, self.h.conv_2_nb)
        #conv2 = self._create_conv(conv1, shape, relu=True, max_poo
ling=False, padding='VALID')
        conv1 = tf.nn.dropout(conv1, keep_prob=conv_2_dropout)

        # Create the first capsules layer
        caps1 = conv_caps_layer(
            input_layer=conv1,
            capsules_size=self.h.caps_1_vec_len,
            nb_filters=self.h.caps_1_nb_filter,
            kernel=self.h.caps_1_size)
        # Create the second capsules layer used to predict the outp
ut
        caps2 = fully_connected_caps_layer(
            input_layer=caps1,
            capsules_size=self.h.caps_2_vec_len,
            nb_capsules=self.NB_LABELS,
            iterations=self.h.routing_steps)

        return caps1, caps2

    def _build_decoder(self, caps2, one_hot_labels, batch_size):
        """
            Build the decoder part from the last capsule layer
            **input:
                *Caps2:  Output of second Capsule layer
                *one_hot_labels
                *batch_size
        """
        labels = tf.reshape(one_hot_labels, (-1, self.NB_LABELS, 1)
)
        # squeeze(caps2):   [?, len_v_j,    capsules_nb]
        # labels:           [?, NB_LABELS,  1] with capsules_nb ==
NB_LABELS
        mask = tf.matmul(tf.squeeze(caps2), labels, transpose_a=Tru
e)
        # Select the good capsule vector
        capsule_vector = tf.reshape(mask, shape=(batch_size, self.h
.caps_2_vec_len))
```

```
        # capsule_vector: [?, len_v_j]

        # Reconstruct image
        fc1 = tf.contrib.layers.fully_connected(capsule_vector, num
_outputs=400)
        fc1 = tf.reshape(fc1, shape=(batch_size, 5, 5, 16))
        upsample1 = tf.image.resize_nearest_neighbor(fc1, (8, 8))
        conv1 = tf.layers.conv2d(upsample1, 4, (3,3), padding='same
', activation=tf.nn.relu)

        upsample2 = tf.image.resize_nearest_neighbor(conv1, (16, 16
))
        conv2 = tf.layers.conv2d(upsample2, 8, (3,3), padding='same
', activation=tf.nn.relu)

        upsample3 = tf.image.resize_nearest_neighbor(conv2, (32, 32
))
        conv6 = tf.layers.conv2d(upsample3, 16, (3,3), padding='sam
e', activation=tf.nn.relu)

        # 3 channel for RGG
        logits = tf.layers.conv2d(conv6, 3, (3,3), padding='same',
activation=None)
        decoded = tf.nn.sigmoid(logits, name='decoded')
        tf.summary.image('reconstruction_img', decoded)

        return decoded

    def init(self):
        """
            Init the graph
        """
        # Get graph inputs
        self.tf_images, self.tf_labels = self._build_inputs()
        # Dropout inputs
        self.tf_conv_2_dropout = tf.placeholder(tf.float32, shape=(
), name='conv_2_dropout')
        # Dynamic batch size
        batch_size = tf.shape(self.tf_images)[0]
        # Translate labels to one hot array
        one_hot_labels = tf.one_hot(self.tf_labels, depth=self.NB_L
ABELS)
        # Create the first convolution and the CapsNet
        self.tf_caps1, self.tf_caps2 = self._build_main_network(sel
f.tf_images, self.tf_conv_2_dropout)

        # Build the images reconstruction
        self.tf_decoded = self._build_decoder(self.tf_caps2, one_ho
t_labels, batch_size)

        # Build the loss
        _loss = self._build_loss(
            self.tf_caps2, one_hot_labels, self.tf_labels, self.tf_
decoded, self.tf_images)
        (self.tf_loss_squared_rec, self.tf_margin_loss_sum, self.tf
_predicted_class,
```

```
            self.tf_correct_prediction, self.tf_accuracy, self.tf_loss
, self.tf_margin_loss,
            self.tf_reconstruction_loss) = _loss

        # Build optimizer
        optimizer = tf.train.AdamOptimizer(learning_rate=self.h.lea
rning_rate)
        self.tf_optimizer = optimizer.minimize(self.tf_loss, global
_step=tf.Variable(0, trainable=False))

        # Log value into tensorboard
        tf.summary.scalar('margin_loss', self.tf_margin_loss)
        tf.summary.scalar('accuracy', self.tf_accuracy)
        tf.summary.scalar('total_loss', self.tf_loss)
        tf.summary.scalar('reconstruction_loss', self.tf_reconstruc
tion_loss)

        self.tf_test = tf.random_uniform([2], minval=0, maxval=None
, dtype=tf.float32, seed=None, name="tf_test")

        self.init_session()


    def _build_loss(self, caps2, one_hot_labels, labels, decoded, i
mages):
        """
            Build the loss of the graph
        """
        # Get the length of each capsule
        capsules_length = tf.sqrt(tf.reduce_sum(tf.square(caps2), a
xis=2, keep_dims=True))

        max_l = tf.square(tf.maximum(0., 0.9 - capsules_length))
        max_l = tf.reshape(max_l, shape=(-1, self.NB_LABELS))
        max_r = tf.square(tf.maximum(0., capsules_length - 0.1))
        max_r = tf.reshape(max_r, shape=(-1, self.NB_LABELS))
        t_c = one_hot_labels
        m_loss = t_c * max_l + 0.5 * (1 - t_c) * max_r
        margin_loss_sum = tf.reduce_sum(m_loss, axis=1)
        margin_loss = tf.reduce_mean(margin_loss_sum)

        # Reconstruction loss
        loss_squared_rec = tf.square(decoded - images)
        reconstruction_loss = tf.reduce_mean(loss_squared_rec)

        # 3. Total loss
        loss = margin_loss + (0.0005 * reconstruction_loss)

        # Accuracy
        predicted_class = tf.argmax(capsules_length, axis=1)
        predicted_class = tf.reshape(predicted_class, [tf.shape(cap
sules_length)[0]])
        correct_prediction = tf.equal(predicted_class, labels)
        accuracy = tf.reduce_mean(tf.cast(correct_prediction, tf.fl
oat32))
```

```
        return (loss_squared_rec, margin_loss_sum, predicted_class,
correct_prediction, accuracy,
                loss, margin_loss, reconstruction_loss)

    def optimize(self, images, labels, tb_save=True):
        """
            Train the model
            **input: **
                *images: Image to train the model on
                *labels: True classes
                *tb_save: (Boolean) Log this optimization in tensor
board
            **return: **
                Loss: The loss of the model on this batch
                Acc: Accuracy of the model on this batch
        """
        tensors = [self.tf_optimizer, self.tf_margin_loss, self.tf_
accuracy, self.tf_tensorboard]
        _, loss, acc, summary = self.sess.run(tensors,
            feed_dict={
            self.tf_images: images,
            self.tf_labels: labels,
            self.tf_conv_2_dropout: self.h.conv_2_dropout
        })

        if tb_save:
            # Write data to tensorboard
            self.train_writer.add_summary(summary, self.train_write
r_it)
            self.train_writer_it += 1

        return loss, acc

    def evaluate(self, images, labels, tb_train_save=False, tb_test
_save=False):
        """
            Evaluate dataset
            **input: **
                *images: Image to train the model on
                *labels: True classes
                *tb_train_save: (Boolean) Log this optimization in
tensorboard under the train part
                *tb_test_save: (Boolean) Log this optimization in t
ensorboard under the test part
            **return: **
                Loss: The loss of the model on this batch
                Acc: Accuracy of the model on this batch
        """
        tensors = [self.tf_margin_loss, self.tf_accuracy, self.tf_t
ensorboard]
        loss, acc, summary = self.sess.run(tensors,
                feed_dict={
                self.tf_images: images,
                self.tf_labels: labels,
                self.tf_conv_2_dropout: 1.
            })
```

```
        if tb_test_save:
            # Write data to tensorboard
            self.test_writer.add_summary(summary, self.test_writer_
it)
            self.test_writer_it += 1

        if tb_train_save:
            # Write data to tensorboard
            self.train_writer.add_summary(summary, self.train_write
r_it)
            self.train_writer_it += 1

        return loss, acc

    def predict(self, images):
        """
            Method used to predict a class
            Return a softmax
            **input: **
                *images: Image to train the model on
            **return:
                *softmax: Softmax between all capsules
        """
        tensors = [self.tf_caps2]

        caps2 = self.sess.run(tensors,
            feed_dict={
            self.tf_images: images,
            self.tf_conv_2_dropout: 1.
        })[0]

        # tf.sqrt(tf.reduce_sum(tf.square(caps2), axis=2, keep_dims
=True))
        caps2 = np.sqrt(np.sum(np.square(caps2), axis=2, keepdims=T
rue))
        caps2 = np.reshape(caps2, (len(images), self.NB_LABELS))
        # softmax
        softmax = np.exp(caps2) / np.sum(np.exp(caps2), axis=1, kee
pdims=True)

        return softmax

    def reconstruction(self, images, labels):
        """
            Method used to get the reconstructions given a batch
            Return the result as a softmax
            **input: **
                *images: Image to train the model on
                *labels: True classes
        """
        tensors = [self.tf_decoded]

        decoded = self.sess.run(tensors,
            feed_dict={
            self.tf_images: images,
```

```
            self.tf_labels: labels,
            self.tf_conv_2_dropout: 1.
        })[0]

        return decoded

    def evaluate_dataset(self, images, labels, batch_size=10):
        """
            Evaluate a full dataset
            This method is used to fully evaluate the dataset batch
per batch. Useful when
            the dataset can't be fit inside to the GPU.
            *input: **
                *images: Image to train the model on
                *labels: True classes
            *return: **
                *loss: Loss overall your dataset
                *accuracy: Accuracy overall your dataset
                *predicted_class: Predicted class
        """
        tensors = [self.tf_loss_squared_rec, self.tf_margin_loss_su
m, self.tf_correct_prediction,
                   self.tf_predicted_class]

        loss_squared_rec_list = None
        margin_loss_sum_list = None
        correct_prediction_list = None
        predicted_class = None

        b = 0
        for batch in self.get_batches([images, labels], batch_size,
shuffle=False):
            images_batch, labels_batch = batch
            loss_squared_rec, margin_loss_sum, correct_prediction,
classes = self.sess.run(tensors,
                feed_dict={
                self.tf_images: images_batch,
                self.tf_labels: labels_batch,
                self.tf_conv_2_dropout: 1.
            })
            if loss_squared_rec_list is not None:
                predicted_class = np.concatenate((predicted_class,
classes))
                loss_squared_rec_list = np.concatenate((loss_square
d_rec_list, loss_squared_rec))
                margin_loss_sum_list = np.concatenate((margin_loss_
sum_list, margin_loss_sum))
                correct_prediction_list = np.concatenate((correct_p
rediction_list, correct_prediction))
            else:
                predicted_class = classes
                loss_squared_rec_list = loss_squared_rec
                margin_loss_sum_list = margin_loss_sum
                correct_prediction_list = correct_prediction
            b += batch_size
```

```
        margin_loss = np.mean(margin_loss_sum_list)
        reconstruction_loss = np.mean(loss_squared_rec_list)
        accuracy = np.mean(correct_prediction_list)

        loss = margin_loss

        return loss, accuracy, predicted_class
```

EK 3 CapsNet Modelinde Kullanılan “model_base.py” Dosyası

Bu çalışmada CapsNet modelinin gerçeklenmesi için https://github.com/thibo73800/capsnet-traffic-sign-classifier adresindeki Github deposundan faydalanılmıştır.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import tensorflow as tf
from collections import Counter
from utils import Utils as U
import json
import numpy as np
from logger import Logger
import time
import pickle
import os

log = Logger("ModelBase")

class Hyperparameters(object):
    """
        Simple class used to store Hyperparameters
    """
    def __init__(self):
        super(Hyperparameters, self).__init__()
        # List used to store list of hyperparameters name
        self.hyp_list = []

    def set_hyp(self, hyp):
        """
            Method used to store hyperparameters inside this class
            **input: **
                *hyp (Dict) Dictionary storing all hyperparameters
values
        """
        for key in hyp:
            self.hyp_list.append(key)
            setattr(self, key, hyp[key])

class ModelBase(object):
    """
        Base Model Class
    """

    #  Hyp : Hyperparameters
    DEFAULT_OUTPUT = "outputs"
    DEFAULT_CHECKPOINT_FOLDER = "checkpoints"
```

```
    def __init__(self, model_name, hyperparameters_name=None, hyper
parameters_content=None, output_folder=None):
        """
            **input:
                *hyperparameters_name: [Optional] (String|None) Pat
h to the hyperparameters file
                                       By default: hyperparameters.
json
                *model_name: (Integer) Name of this model
        """
        super(ModelBase, self).__init__()

        self.current_dir = os.path.dirname(os.path.realpath(__file_
_))
        # Output folder
        if output_folder is None:
            self.output_folder = os.path.join(
                os.path.dirname(os.path.abspath(__file__)), self.DE
FAULT_OUTPUT)
        else:
            self.output_folder = output_folder

        hyp_folder = "settings"
        hyp_filename = "hyperparameters.json"
        hyp_path = os.path.join(self.current_dir, os.path.join(hyp_
folder, hyp_filename))
        self.checkpoints_folder = os.path.join(self.output_folder,
self.DEFAULT_CHECKPOINT_FOLDER)

        # Set hyperparameters path
        if hyperparameters_name is not None:
            hyp_path = os.path.join(
                self.current_dir, os.path.join(hyp_folder, hyperpar
ameters_name))
        hyp_path = hyp_path if hyperparameters_name is None else hy
p_path
        # Load hyperparameters content
        if hyperparameters_content is None:
            hyp_content = U.read_json_file(hyp_path)
        else:
            hyp_content = hyperparameters_content
        # Set hyperparameters
        self.h = Hyperparameters()
        self.h.set_hyp(hyp_content)
        # Set model names
        self.name = model_name
        self.model_name = model_name
        self._set_hyperparameters_name()
        # Since hyperparameters had changed, we need to set again e
ach name
        self._set_names()

    def _create_conv(self, prev, shape, padding='VALID', strides=[1
, 1, 1, 1], relu=False,
                     max_pooling=False, mp_ksize=[1, 2, 2, 1], mp_s
trides=[1, 2, 2, 1]):
```

```
        """
            Create a convolutional layer with relu and/mor max pool
ing(Optional)
        """
        conv_w = tf.Variable(tf.truncated_normal(shape=shape, mean
= 0, stddev = 0.1,  seed=0))
        conv_b = tf.Variable(tf.zeros(shape[-1]))
        conv   = tf.nn.conv2d(prev, conv_w, strides=strides, paddin
g=padding) + conv_b

        if relu:
            conv = tf.nn.relu(conv)

        if max_pooling:
            conv = tf.nn.max_pool(conv, ksize=mp_ksize, strides=mp_
strides, padding='VALID')

        return conv

    def _fc(self, prev, input_size, output_size, relu=False, sigmoi
d=False, no_bias=False,
            softmax=False):
        """
            Create fully connecter layer with relu(Optional)
        """
        fc_w = tf.Variable(
            tf.truncated_normal(shape=(input_size, output_size), me
an = 0., stddev = 0.1))
        fc_b = tf.Variable(tf.zeros(output_size))
        pre_activation = tf.matmul(prev, fc_w)
        activation = None

        if not no_bias:
            pre_activation = pre_activation + fc_b
        if relu:
            activation = tf.nn.relu(pre_activation)
        if sigmoid:
            activation = tf.nn.sigmoid(pre_activation)
        if softmax:
            activation = tf.nn.softmax(pre_activation)

        if activation is None:
            activation = pre_activation

        return activation, pre_activation

    def init_session(self):
        """
            Init tensorflow session
            A saver property is create at the same time
        """
        #  Create session
        self.saver = tf.train.Saver()
        self.sess = tf.Session()
        # Init variables
        self.sess.run(tf.global_variables_initializer())
```

```
        # Tensorboard
        self.tf_tensorboard = tf.summary.merge_all()
        train_log_name = os.path.join(
            os.path.join(self.output_folder, "tensorboard"), self.n
ame, self.sub_train_log_name)
        test_log_name = os.path.join(
            os.path.join(self.output_folder, "tensorboard"), self.n
ame, self.sub_test_log_name)
        self.train_writer = tf.summary.FileWriter(train_log_name, s
elf.sess.graph)
        self.test_writer = tf.summary.FileWriter(test_log_name)
        self.train_writer_it = 0
        self.test_writer_it = 0

        # Backup tensors
        backup_tensors = {}
        for field in dir(self):
            if "tf_" in field and field.index("tf_") == 0:
                backup_tensors[field] = getattr(self, field).name
        tf.constant(json.dumps(backup_tensors), dtype=tf.string, na
me="model_base_tensors_backup")
        # Backup hyperparameters
        backup_hyp = {}
        for field in self.h.hyp_list:
            value = getattr(self.h, field)
            d_type = tf.int32 if isinstance(value, int) else tf.flo
at32
            n_cst = tf.constant(value, dtype=d_type, name="hyp/%s"
% field)
            backup_hyp[field] = n_cst.name
        tf.constant(json.dumps(backup_hyp), dtype=tf.string, name="
model_base_hyp_backup")

    def get_equal_batches(self, data, labels, batch_size):
        """
            This method will return a generator class which could b
e used to
            get new batches with the same number of rows for each c
lass

            **input:**
                *batch_size (int) Size of each batch
             **return (Python Generator of Batch class)**
        """
        labels = np.array(labels)

        indexs = np.arange(len(data))
        np.random.shuffle(indexs)

        data = data[indexs]
        labels = labels[indexs]

        max_size = Counter(labels).most_common()[-1][1]
        unique_label = np.array(list(set(labels)))
        nb_classes = len(unique_label)
```

```
        if batch_size > max_size:
            batch_size = max_size

        batch_per_class = batch_size // nb_classes
        iterations = max_size // batch_per_class

        for it in range(iterations):

            indexes = []

            for label in unique_label:
                n_indexes = np.where(labels==label)[0][it * batch_p
er_class: (it + 1) * batch_per_class]
                n_indexes = n_indexes.tolist()
                indexes += n_indexes

            indexes = np.array(indexes)

            x = data[indexes]
            y = labels[indexes]

            yield x, y


    def get_batches(self, data_list, batch_size, shuffle=True):
        """
            This method will return a generator class which could b
e used to
            get new batches.

            **input:**
                *batch_size (int) Size of each batch
             **return (Python Generator of Batch class)**
        """
        if shuffle:
            indexs = np.arange(len(data_list[0]))
            np.random.shuffle(indexs)

            for d, data in enumerate(data_list):
                data_list[d] = np.array(data_list[d])
                data_list[d] = data_list[d][indexs]

        iterations = len(data_list[0]) // batch_size
        for iteration in range(iterations):
            yield (dt[iteration * batch_size: (iteration + 1) * bat
ch_size] for dt in data_list)

    def save(self, name=None):
        """
            Save the model
        """
        log.info("Saving model ...")

        if name is None:
            name = self.model_name
```

```
        if not os.path.exists(self.checkpoints_folder):
            os.makedirs(self.checkpoints_folder)

        save_path = self.saver.save(
            self.sess, os.path.join(self.checkpoints_folder, name))

        log.info("Model successfully saved here: %s" % save_path)

    def _set_hyperparameters_name(self):
        """
            Convert hyperparameters dict to a string
            This string will be used to set the models names
        """
        # Generate a little name for each hyperparameters
        hyperparameters_names = [("".join([p[0] for p in hyp.split(
"_")]), getattr(self.h, hyp))
                                 for hyp in self.h.hyp_list]
        self.hyperparameters_name = ""
        for index_hyperparameter, hyperparameter in enumerate(hyper
parameters_names):
            short_name, value = hyperparameter
            prepend = "" if index_hyperparameter == 0 else "_"
            self.hyperparameters_name += "%s%s_%s" % (prepend, shor
t_name, value)

    def _set_names(self):
        """
            Set all model names
        """
        name_time = "%s--%s" % (self.model_name, time.time())
        # model_name is used to set the ckpt name
        self.model_name = "%s--%s" % (self.hyperparameters_name, na
me_time)
        # sub_train_log_name is used to set the name of the trainin
g part in tensorboard
        self.sub_train_log_name = "%s-train--%s" % (self.hyperparam
eters_name, name_time)
        # sub_test_log_name is used to set the name of the testing
part in tensorboard
        self.sub_test_log_name = "%s-test--%s" % (self.hyperparamet
ers_name, name_time)

    def dump_batch(self, folder, data):
        """
            Save batches
            Mainly used for Reinforcement Learning
        """
        folder = os.path.join(os.path.dirname(os.path.abspath(__fil
e__)), folder)
        # Create folder if not exist
        if not os.path.exists(folder):
            os.makedirs(folder)

        pickle.dump(data, open(os.path.join(folder, str(time.time()
)), "wb" ))
```

```
    def load(self, ckpt):
        """
            Load a model
        """
        log.info("Loading ckpt ...")
        #loaded_graph = tf.Graph()
        #tf.reset_default_graph()
        #g = tf.Graph()
        #with g.as_default():
        self.sess = tf.Session()
        # Load the graph
        loader = tf.train.import_meta_graph(ckpt + '.meta')
        loader.restore(self.sess, ckpt)

        g = tf.get_default_graph()

        # Search for the backup tensor
        tensor_names = [
            n.name for n in g.as_graph_def().node if "model_base_te
nsors_backup" in n.name]

        # Search for the backup hyp
        hyp_names = [
            n.name for n in g.as_graph_def().node if "model_base_hy
p_backup" in n.name]

        # Get the tensor string
        #tensors = g.get_tensor_by_name(names[0])
        tensors = g.get_operation_by_name(tensor_names[0]).outputs
        hyps = g.get_operation_by_name(hyp_names[0]).outputs

        #self.sess.run(tf.global_variables_initializer())

        tensors = self.sess.run(tensors)[0]
        tensors = json.loads(tensors)
        for tensor in tensors:
            try:
                n_tensor = g.get_tensor_by_name(tensors[tensor])
            except Exception as e:
                n_tensor = g.get_operation_by_name(tensors[tensor])
            setattr(self, tensor, n_tensor)

        hyps = self.sess.run(hyps)[0]
        hyps = json.loads(hyps)
        for hyp in hyps:
            n_hyp = g.get_tensor_by_name(hyps[hyp])
            setattr(self.h, hyp, self.sess.run(n_hyp))

        log.info("Ckpt ready")

        # Tensorboard
        self.tf_tensorboard = tf.summary.merge_all()
        train_log_name = os.path.join(
            os.path.join(self.output_folder, "tensorboard"), self.n
ame, self.sub_train_log_name)
```

```
        test_log_name = os.path.join(
            os.path.join(self.output_folder, "tensorboard"), self.n
ame, self.sub_test_log_name)
        self.train_writer = tf.summary.FileWriter(train_log_name, s
elf.sess.graph)
        self.test_writer = tf.summary.FileWriter(test_log_name)
        self.train_writer_it = 0
        self.test_writer_it = 0

        self.model_name = ckpt.split("/")[-1]
        self.saver = tf.train.Saver()


if __name__ == '__main__':
    base_model = BaseModel("test")
```

EK 4 CapsNet Modelinde Kullanılan “utils.py” Dosyası

Bu çalışmada CapsNet modelinin gerçeklenmesi için https://github.com/thibo73800/capsnet-traffic-sign-classifier adresindeki Github deposundan faydalanılmıştır.

```
# coding: utf-8

import numpy as np
import json
import sys
import os


class Utils(object):
    """
        Util class to store all common method use in this project
    """

    def __init__(self, arg):
        super(Utils, self).__init__()

    @staticmethod
    def progress(count, total, suffix=''):
        """
            Utils method to display a progress bar
            **input: **
                *count: current progression
                *total: Max progress bar length
        """
        bar_len = 60
        filled_len = int(round(bar_len * count / float(total)))

        percents = round(100.0 * count / float(total), 1)
        bar = '=' * filled_len + '-' * (bar_len - filled_len)

        sys.stdout.write('[%s] %s%s ...%s\r' % (bar, percents, '%',
suffix))
        sys.stdout.flush()

    @staticmethod
    def read_json_file(path):
        """
            Utils method to open, read and return a json file conte
nt
            **input: **
                *path: (String) Path to the json file to read
        """
        with open(path, "r") as f:
            json_content = json.loads(f.read())
        return json_content
```

EK 5 CapsNet Modelinde Kullanılan “logger.py” Dosyası

Bu çalışmada CapsNet modelinin gerçeklenmesi için https://github.com/thibo73800/capsnet-traffic-sign-classifier adresindeki Github deposundan faydalanılmıştır.

```
#!/usr/bin/env python
# -*- coding: utf-8 -*-

import logging
from logging.handlers import RotatingFileHandler

logger = logging.getLogger()
logger.setLevel(logging.DEBUG)
formatter = logging.Formatter('%(asctime)s:: %(levelname)s:: %(mess
age)s')
file_handler = RotatingFileHandler('dory_ai.log', 'a', 1000000, 1)
file_handler.setLevel(logging.INFO)
file_handler.setFormatter(formatter)
logger.addHandler(file_handler)
stream_handler = logging.StreamHandler()
stream_handler.setLevel(logging.DEBUG)
logger.addHandler(stream_handler)


class Logger(object):

    def __init__(self, label):
        super(Logger, self).__init__()
        self.label = label
        self.logger = logger

    def debug(self, string):
        self.logger.debug("%s::%s" % (self.label, string))

    def info(self, string):
        self.logger.info("%s::%s" % (self.label, string))

    def warning(self, string):
        self.logger.warning("%s::%s" % (self.label, string))

    def error(self, string):
        self.logger.error("%s::%s" % (self.label, string))

    def critical(self, string):
        self.logger.critical("%s::%s" % (self.label, string))
```

ÖZGEÇMİŞ

Adı Soyadı : Korhan Mutludoğan
Doğum Yeri ve Tarihi : Bursa – 10.01.1993
Yabancı Dil : İngilizce

Eğitim Durumu
Lise : Bursa Anadolu Lisesi, 2011
Lisans : İstanbul Üniversitesi Bilgisayar Mühendisliği, 2017

Çalıştığı Kurum/Kurumlar : Ne-Ka Elektronik San. Tic. Ltd. Şti.

İletişim (e-posta) : korhanmd@gmail.com

Yayınları :

Bilgin, M., Mutludoğan, K. 2019. American sign language character recognition with capsule networks. 3rd International Symposium on Multidisciplinary Studies and Innovative Technologies, 11-13 Ekim 2019, Ankara, Türkiye.