KOCAELİ ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ BÖLÜMÜ

PROJE B

DERİN ÖĞRENME İLE TRAFİK LEVHA SINIFLANDIRMASI

ASLI BOZKURT

KOCAELİ 2024

KOCAELİ ÜNİVERSİTESİ
TEKNOLOJİ FAKÜLTESİ

BİLİŞİM SİSTEMLERİ MÜHENDİSLİĞİ BÖLÜMÜ

PROJE B

DERİN ÖĞRENME İLE TRAFİK LEVHA SINIFLANDIRMASI

ASLI BOZKURT

Doç. Dr. ,Süleyman EKEN

Danışman, Kocaeli Üniv. **…….**

Tezin Savunulduğu Tarih: 15.05.2024

ÖNSÖZ VE TEŞEKKÜR

Bu tez çalışması trafik işaretlerinin makine öğrenimi metoduyla tanınabilmesi için CNN modeller kullanılmış olup ResNet50, CNN ve VGG16 modellerinde yüksek test başarısı sağlamak amacıyla gerçekleştirilmiştir.

Tez çalışmam süresince destekleriyle katkıda bulunan hocam Süleyman Eken'e teşekkür ediyorum.

Hayatım boyunca bana güç veren en büyük destekçilerim, her aşamada sıkıntılarımı ve mutluluklarımı paylaşan sevgili aileme teşekkürlerimi sunarım.

Mayıs– 2024 Aslı BOZKURT

Bu dokümandaki tüm bilgiler, etik ve akademik kurallar çerçevesinde elde edilip sunulmuştur. Ayrıca yine bu kurallar çerçevesinde kendime ait olmayan ve kendimin üretmediği ve başka kaynaklardan elde edilen bilgiler ve materyaller (text, resim, şekil, tablo vb.) gerekli şekilde referans edilmiş ve dokümanda belirtilmiştir.

Öğrenci No: 201307025

Adı Soyadı: Aslı BOZKURT

İmza: 

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

DERİN ÖĞRENME İLE TRAFİK LEVHA SINIFLANDIRMASI

ÖZET

Günümüzün teknolojik koşulları altında, araçlarda sürücü destek sistemlerinin kullanımı oldukça yaygın hale gelmiştir. Araç sayısındaki artış nedeniyle meydana gelebilecek kazaların önlenmesinde, bu destek sistemlerinin önemli bir rol oynaması hedeflenmektedir. Sürücü destek sistemlerinin en önde gelenlerinden biri, yollardaki trafik işaretlerini tanıyarak sürücüye yol koşulları hakkında bilgi veren otomatik trafik işareti tanıma sistemleridir. Trafik kurralarının uyulması hayati bir önem sahiptir.

Bu tezin amacı yapay zekanın temellerini kullanarak trafik levhalarını doğru bir şekilde tanıyabilmesi için derin öğrenmeye dayalı modeler ile çalışılmıştır. Bu modellerin başarıları ve test sonuçları ele alınmıştır.

Bu çalışmada 2716 adet veri ve 14 sınıf ile çalışılmıştır. CNN modeler kullanılmıştır. Temel yapı CNN, VGG16 ve Resnet50 modelleriyle çalışılmıştır. En iyi performans gösteren CNN modellerden VGG16 %99 başarı ve test sonuçlarını vermiştir.

ANAHTAR KELİMELER: Derin Öğrenme, Makine Öğrenmesi, Trafik İşareti, Evrişimli Sinir Ağları, CNN, VGGNET16, ResNet50

TRAFFIC SIGN CLASSIFICATION WITH DEEP LEARNING

ABSTRACT

Under today's technological conditions, the use of driver assistance systems in vehicles has become quite common. It is aimed that these support systems will play an important role in preventing accidents that may occur due to the increase in the number of vehicles. One of the most prominent driver assistance systems is automatic traffic sign recognition systems, which recognize traffic signs on the roads and provide information to the driver about road conditions. Compliance with traffic rules is of vital importance.

The aim of this thesis is to work with deep learning-based models to accurately recognize traffic signs using the foundations of artificial intelligence. The success and test results of these models are discussed.

In this study, 2716 data and 14 classes were studied. CNN models were used. The basic structure was worked with CNN, VGG16 and Resnet50 models. VGG16, one of the best performing CNN models, gave 99% success and test results.

KEYWORDS: Deep Learning, Machine Learning, Traffic Sign, Convolutional Neural Networks, CNN, VGGNET16, ResNet50

GİRİŞ

Otomobil teknolojisi, bilgisayarların ve elektronik sistemlerin gelişimiyle birlikte mekanik sistemlerden elektronik sistemlere doğru evrilmiştir. Elektronik sistemler başlangıçta, araçların yol tutuşunu artıran, fren mesafesini azaltan ve sürüş konforunu sağlayan çözümler olarak kullanılmıştır. Teknolojinin ilerlemesiyle birlikte, bu sistemler daha da gelişmiş ve araçlar, otonom sürüş teknolojisinin temelini oluşturan birçok yardımcı özelliği kazanmıştır. Bu özellikler arasında paralel park etme, adaptif hız ve far kontrolü, sürücü takip sistemi, trafik işareti tanıma sistemi, şerit takip sistemi ve otomatik frenleme ile çarpışma önleyici sistemler bulunmaktadır. Bu yenilikler, sürüş konforu, can ve mal güvenliği ile ekonomik verimlilik açısından büyük önem taşımaktadır.

Trafik işaretleri, sürüş güvenliğini sağlamak amacıyla hayati bir öneme sahiptir. Sürücüler, güvenli bir sürüş için trafik işaretlerini dikkatlice takip etmeli ve bu işaretlere göre uygun reaksiyonlar göstermelidir. Örneğin, "Sola Tehlikeli Viraj" veya azami hız sınırı işaretleri gibi uyarılar, sürücüler tarafından fark edilmeli ve buna göre hız azaltılmalıdır. Ancak, dikkat dağınıklığı veya çevresel etkenler nedeniyle trafik işaretlerinin zamanında fark edilmesi her zaman mümkün olmayabilir. Bu tür olumsuz durumları önlemek amacıyla, sürücüleri trafik işaretleri konusunda önceden uyaran sistemler günümüz araç teknolojilerinde önemli bir yer tutmaktadır.

Modern araçlar, üzerlerindeki kameralar sayesinde yol taraması yaparak trafik işaretlerini otomatik olarak tanıyabilir hale gelmiştir. Düşük seviye otonom sistemler, sınırlı sayıda trafik işaretini tanıyabilir. Örneğin, azami hız limiti tabelalarını tanıyarak, sürücünün izin verilen hızı aşması durumunda sesli uyarılar yapabilen sistemler bulunmaktadır. Sürücü izin verirse, araç bu hız limitini aşmayı da engelleyebilir.

Daha yüksek seviye otonom sistemler ise, yalnızca azami hız limitlerini değil, aynı zamanda daha fazla trafik işaretini, yol çizgilerini ve çevresel faktörleri taramak

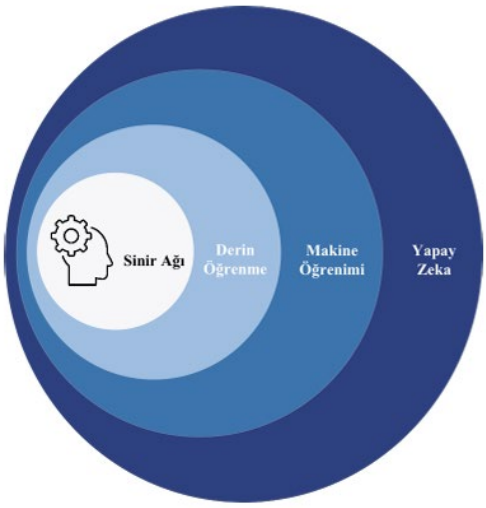
zorundadır. Örneğin, bir aracın şerit değiştirme kararı verebilmesi için, trafik işaretlerini ve yol çizgilerini tanıması ve geçeceği şeritteki trafik durumunu değerlendirerek karar vermesi gerekmektedir.

Bu tez çalışmasında, otonom sürüş seviyelerinin temelini oluşturan trafik işaretlerinin makine öğrenmesinin bir alt dalı olan derin öğrenme metoduyla tanınması hedeflenmiştir.

1. MAKİNE ÖĞRENİMİ

Makine öğrenmesi, günümüzde birçok sektörde (bankacılık, eğlence, finans, biyoteknoloji, eğitim, oyun, pazarlama vb.) yaygın olarak kullanılmaktadır. Bu teknolojinin temel avantajı, büyük ve karmaşık veri yığınlarından anlamlı çıkarımlar yapabilme yeteneğidir. İnsanların kısa sürede yorumlayamayacağı veri hacimlerini analiz ederek geleceğe yönelik tahminlerde bulunabilir.

Büyük veri ile çalışan makine öğrenmesi, verilerdeki kalıpları ve korelasyonları tespit ederek en iyi kararları ve tahminleri yapar. Makine öğrenmesi, yapay zekanın alt dalıdır. Derin öğrenmenin üst kümesinde yer alır; sinir ağları, derin öğrenme ve makine öğrenmesi, yapay zekânın bileşenleridir. Bu sistemler, zamanla daha fazla veri ile beslendikçe doğruluklarını artırırlar.



Şekil 1.1. Makine Öğrenimi Şeması

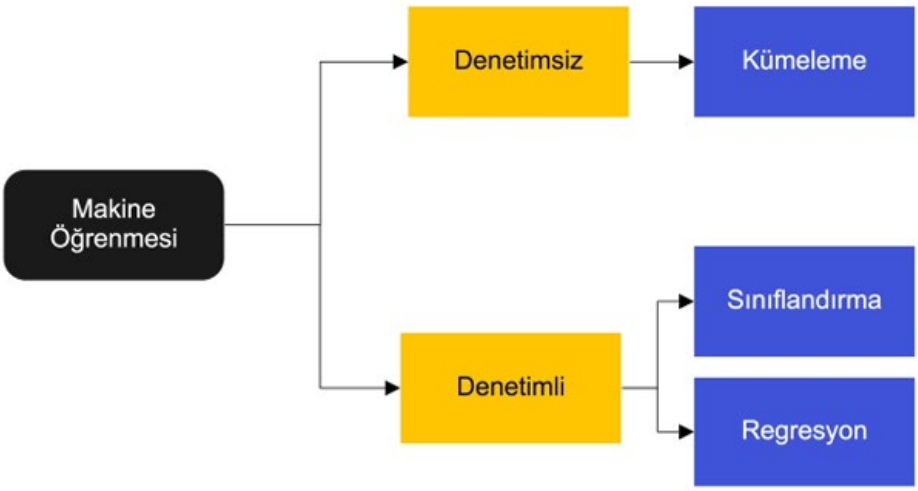
Makine öğrenimi, yapay zekânın bir alt dalı olup, verilerden öğrenerek tahminler ve sınıflandırmalar yapan algoritmalar geliştirmeyi amaçlar. Bu süreçte, büyük veri setleri kullanıldıkça, algoritmaların doğruluğu ve kesinliği artar. Makine öğrenimi terimi ilk olarak 1959'da Arthur Lee Samuel tarafından, IBM için dama oyunu tasarlarken kullanılmıştır. Samuel, makine öğrenimini "bilgisayarların açıkça programlanmadan öğrenmesini sağlayan bir araştırma alanı" olarak tanımlamıştır. 1962'de IBM7094 bilgisayarına karşı dama oyununu kaybeden Robert Nealey, makine öğreniminin gücünü ilk elden deneyimlemiştir. Samuel, dama oyunu için minimax algoritması ve ezberci öğrenme yöntemleri kullanarak, bilgisayarın gelecekteki hamleleri tahmin etme ve öğrenme kapasitesini artırmıştır.

Şekil1.2. Makine Öğrenimi Çalışma Prensibi

Geleneksel programlamada, programcı tarafından belirlenen kurallar (IF-THEN yapısı) kullanılarak veriler işlenir ve belirli çıktılar elde edilir. Ancak, makine öğreniminde insan müdahalesi minimaldir. Algoritmalar, verilerden beslenerek ve geçmiş gözlemleri değerlendirerek kendi başlarına öğrenir ve karar verirler. Yapay zekâ, makine öğreniminden farklı olarak, makinelerin yeni beceriler öğrenmesini ve sorunları insan benzeri bir şekilde çözmesini sağlar. Makine öğrenimi ise verilerden öğrenme sürecine odaklanır ve etiketlenmiş verilerle eğitilen algoritmalar, yeni verileri sınıflandırmayı otomatik olarak öğrenir. Örneğin, bir algoritma elma ve armutları etiketlemeyi öğrendikten sonra, gelecekteki meyve tanımlamalarını kendi başına yapabilir.

1.1. Makine Öğrenimi Grupları

Makine öğrenimi, makinelerin karar verirken kullandığı kurallar dizisi ve çeşitli algoritmalar aracılığıyla farklı yöntemlerle sınıflandırılır. Temelde, makine öğrenimi yöntemleri denetimli ve denetimsiz öğrenme olarak ikiye ayrılır. Ancak, zamanla bu sınıflara yarı denetimli ve pekiştirmeli öğrenme de eklenmiştir.



Şekil 1.3. Makine Öğrenimi Grupları

1.1.1.Denetimli Öğrenme (Supervised Learning)

Denetimli öğrenme, etiketlenmiş verileri kullanarak çalışan bir makine öğrenimi türüdür. Bu türde, her giriş verisi, doğru çıktılarla eşleştirilmiş şekilde bir veri kümesinde yer alır. Bu doğru çıktılar, algoritmanın öğrenme sürecinde kullanılır ve modelin yeni, henüz görmediği giriş verileri için doğru tahminler yapmasına yardımcı olur. Denetimli öğrenme ile geliştirilen modellerin amacı, eğitim sürecindeki verilerden öğrendiği desenleri genelleyerek yeni verilere uygulamaktır.

Denetimli öğrenme algoritmaları iki ana kategoride incelenir. Regresyon problemleri için yaygın olarak Doğrusal Regresyon ve Destek Vektör Regresyonu (SVR) kullanılır. Örneğin, bir evin özelliklerine göre fiyat tahmini yapmak regresyon problemidir.

Sınıflandırma problemleri için ise çeşitli algoritmalar kullanılır. Destek Vektör Makineleri (SVM), verileri sınıflandırmak için en iyi ayrım çizgisini bulur. Karar Ağaçları, veriyi farklı dallara ayırarak sınıflandırma yapar. K-En Yakın Komşu (KNN), yeni bir veriyi, en yakın komşularına dayanarak sınıflandırır. Yapay Sinir Ağları (ANN), çok katmanlı yapısı ile karmaşık desenleri öğrenip sınıflandırabilir.

Yapay Sinir Ağları, karmaşık ilişkileri tanımlama ve geniş bir uygulama yelpazesine sahip olma potansiyeliyle güçlü bir makine öğrenimi modelidir. Ancak, büyük veri setleri ve karmaşık yapılarla çalışırken dikkatli bir şekilde ele alınmalı ve uygun önlemler alınmalıdır.

Bu algoritmalar, denetimli öğrenmede çeşitli problemleri çözmek için kullanılır ve her biri, farklı veri türleri ve problemlere göre özelleştirilmiştir. Denetimli öğrenmenin gücü, doğru etiketlenmiş verilerle beslenen algoritmaların, yeni ve bilinmeyen veriler üzerinde yüksek doğrulukla tahmin yapabilmesinden gelir.

1.1.2.Denetimsiz Öğrenme (Unsupervised Learning)

Denetimsiz Öğrenme (Unsupervised Learning), etiketlenmemiş veri kümeleriyle çalışan bir makine öğrenimi türüdür. Bu türde, sadece giriş verilerinin olduğu ve herhangi bir etiketin bulunmadığı veri kümesi kullanılır. Algoritmanın temel amacı, veri kümesindeki yapıları ve desenleri keşfetmektir. Bu süreçte, etiketlenmemiş verilerdeki iç görüleri ve ilişkileri ortaya çıkarır.

Denetimsiz öğrenme, eğitim sırasında etiketsiz verilerin kullanılmasına dayanır. Bu veriler, sistem tarafından anlaşılmaya, tanımlanmaya ve keşfedilmeye çalışılır. Bu nedenle, sisteme beklenen sonuçlar önceden bilinmez. Denetimsiz öğrenme modeli doğru cevaplarla eğitilmez; bunun yerine, kendi başına veri içindeki kalıpları bulması beklenir.

Bu model, gruplama yaparak benzerlik veya farklılıkları ortaya çıkarır ve veri setindeki desenleri belirlemeye çalışır. Denetimsiz öğrenme, veri setindeki gizli yapıları ve ilişkileri keşfetmek için kullanılır ve genellikle veri analizi ve veri keşfi gibi alanlarda kullanılır.

2. DERİN ÖĞRENME

Yapay zekanın ilk dönemlerinde, bilgisayarlar biçimsel ve matematiksel kurallarla tanımlanabilen, insanlar için zor ama bilgisayarlar için nispeten basit olan problemleri hızla çözebilmiştir. Ancak, yapay zeka için asıl zorluğun, insanların sezgisel olarak kolayca çözdüğü fakat matematiksel olarak tanımlaması zor olan görevlerde olduğu ortaya çıkmıştır. Bu görevler arasında resimlerdeki kelimeleri,

yüzleri veya nesneleri tanımak gibi duyusal problemler yer alır (Goodfellow ve diğerleri, 2018).

Makine öğrenimi, genellikle yapay zekanın bir alt disiplini olarak kabul edilir ve bu alandaki en etkili teknik olarak öne çıkar. Günümüzde, endüstri ve toplum üzerinde büyük etkiler yaratma potansiyeline sahip araçlar sunarak önemli bir vaatte bulunmaktadır . Makine öğrenimi, yapay zekanın temel fikirlerini alır ve bu fikirleri, sinir ağları kullanarak gerçek dünyadaki problemleri çözmeye odaklanır. Derin öğrenme olarak bilinen daha ileri düzey makine öğrenimi ise, bu araçların ve tekniklerin daha dar bir alt kümesine odaklanır ve düşünme gerektiren hemen hemen her sorunu çözmeyi amaçlar (Goodfellow ve diğerleri, 2018).

Derin öğrenme, karar almak için sinir ağlarını oluşturan ve eğiten özel bir yaklaşımdır. Bir algoritma, girdi verilerini çıktı verilmeden önce bir dizi doğrusal olmayan katmanlı dönüşümden geçiriyorsa, bu derin olarak kabul edilir. Bu yöntem, bilgisayarların deneyimlerden öğrenmesine ve dünyayı kavramlar hiyerarşisi açısından anlamasına olanak tanır. Her kavram, daha basit kavramlarla olan ilişkileri aracılığıyla tanımlanır. Bu yaklaşım, insan operatörlerin bilgisayara ihtiyaç duyduğu tüm bilgileri resmi olarak belirtme gerekliliğini ortadan kaldırır. Kavram hiyerarşisi sayesinde, bilgisayar daha basit kavramlar üzerinden karmaşık kavramları öğrenebilir (Goodfellow ve diğerleri, 2018).

2.1. Yapay Sinir Ağları

2.1.1. Yapay Sinir Ağlarının Kısa Tarihi

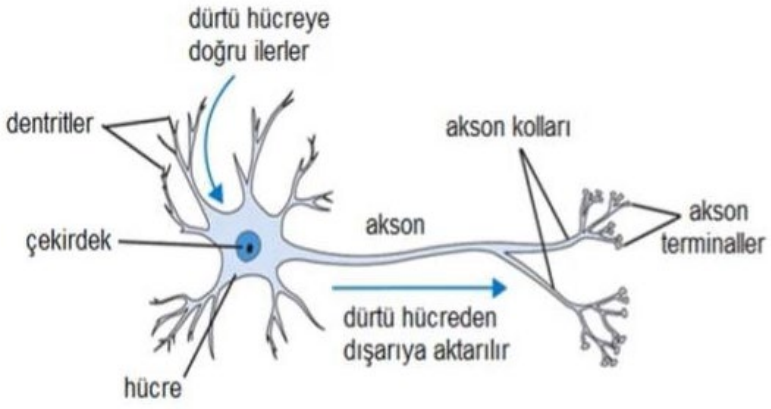
Yapay sinir ağları ve yapay zeka terimleri günümüzde sıkça kullanılsa da, bu konular aslında 1950'li yıllarda ortaya çıkmış ve üzerinde çalışılmaya başlanmıştır. Bu alanda önemli bir başlangıç yapmasını sağlayan Alan Turing’in 1950 yılında “Computing Machinery and Intelligence” adlı makalesidir. Bu sorunun ardından, 1955 yılında John McCarthy ilk kez yapay zeka terimini kullanmıştır. 1957'de ise Frank Rosenblatt, bugünkü yapay sinir ağı modellerinin temelini oluşturan perceptronun tanımını yapmıştır . 1969 yılında Marvin Minsky, XOR probleminin tek katmanlı bir ağ yapısıyla çözülemeyeceğini belirterek, çok katmanlı yapay sinir ağı mimarilerinin temelini atmıştır

1987 yılında, yapay sinir ağlarının öğrenme kapasitesini önemli ölçüde artıran geriye yayılım algoritması geliştirilmiştir. Bu algoritma, ağların edindiği bilgiyi güncelleyebilmesi ve tahminleme yeteneklerini geliştirmesi açısından kritik bir yenilik olmuştur. 1998 yılında, derin öğrenmenin temelini oluşturan evrişimli sinir ağları (CNN) kullanılarak, rakamları sınıflandırmada yüksek başarı sağlayan bir model geliştirilmiştir. 2009 yılında, 167 ülkenin iş birliğiyle oluşturulan IMAGENET veri setinin ücretsiz olarak erişime açılması ve CPU ve GPU teknolojilerindeki gelişmeler, derin öğrenme alanında büyük bir ivme kazandırmıştır. Bu gelişmeler, matris çarpımlarının daha hızlı ve kolay yapılabilmesini sağlamıştır.

Bu ilerlemelerle birlikte, birçok teknoloji şirketi yapay zeka alanında önemli yatırımlar yapmış ve bu sayede, o yıllardan günümüze kadar birçok derin öğrenme modeli geliştirilmiştir. Bu modeller, günümüzde çeşitli alanlarda etkili bir şekilde kullanılmaktadır.

2.1.2. Yapay Sinir Ağlarının Biyolojik Temelleri

Biyolojik canlıların sinir hücre yapısı Şekil 2.1.2. ‘de gösteridiği gibi temel olarak dentrit, çekirdek ve aksonlardan oluşmaktadır. Dentritlerden gelen bilgiler çekirdekte işlendikten sonra aksonlar boyunca diğer sinir hücresine bağlanarak aktarım gerçekleşmektedir.

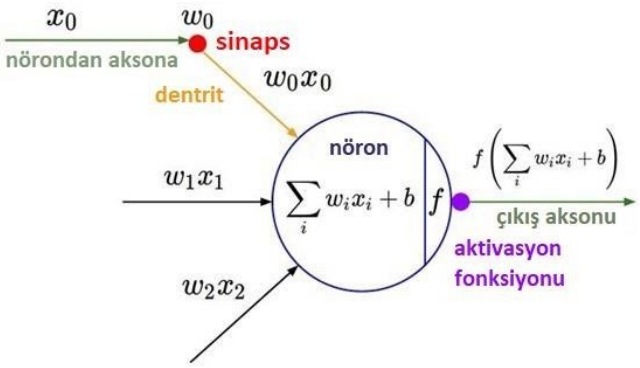


Şekil 2.1. Biyolojik Sinir Hücresi

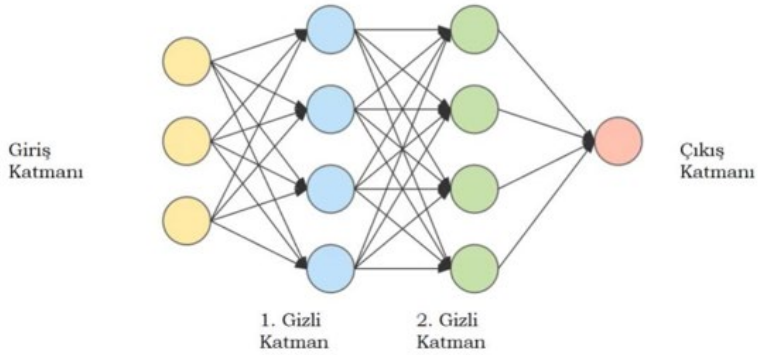
Yapay Sinir Ağları (YSA), biyolojik kardeşlerinin karmaşık yapılarını tam olarak yansıtmasa da, bu doğal örneklerden esinlenerek geliştirilmeye çalışılmıştır. Temel

amacı, beyin öğrenme işlevinin bilgisayarlar tarafından taklit edilebilmesidir. YSA modellerinin geliştirilmesi için ilk adımlar,1943 yılında nörobiyolog W.S. McCulloch ve matematikçi W.A. Pitts tarafından atılmıştır. Modeller; John Hopfield, David Rumelhart gibi araştırmacılar tarafından geliştirilmeye devam etmiştir. Matematiksel olarak modellenmiş hali şekil Şekil 2.2.'de gösterilmiştir. Bu matematiksel modelde, giriş değeri xi (bir önceki nörondan gelen bir değer olabilir) dentrit bölgesinde wi ağırlık değeri ile çarpılır ve bu işlem sinir hücresine iletilir. Sinir hücresinde, dentritten gelen tüm ağırlık ve giriş çarpımları toplanır ve bu toplama bir b değeri eklenir. Ortaya çıkan sonuç, bir aktivasyon

fonksiyonundan geçirilir ve bu sonuç çıkışa aktarılır. Buradaki sonuç,bir nöronun çıkışı olabilir veya başka bir nöronun girişi olabilir. Her bir nöron bu şekilde hesaplandıktan sonra seri ve paralel bağlantılarla diğer nöronlara bağlanır. Bu durum, Şekil 2.3'te gösterilmektedir. Bu bağlantılar, giriş katmanı, gizli katman ve çıkış katmanı olarak adlandırılır. Tek bir giriş ve çıkış katmanı bulunan modeller tek katmanlı, birden fazla gizli katman içeren modeller ise çok katmanlı model olarak adlandırılır.



Şekil 2.2. Sinir Ağlarının Matematiksel Modeli

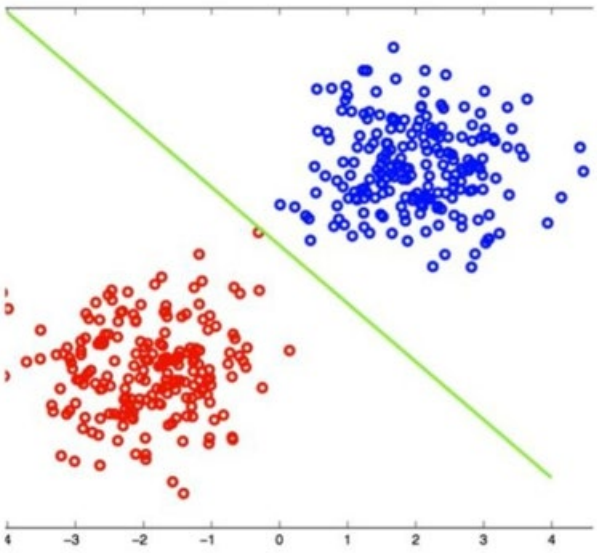


Şekil 2.3. Genel Yapay Sinir Ağı Modeli

2.1.3. Tek Katmanlı Sinir Ağı Modelleri

Tek katmanlı sinir ağlarının basit bir yapısı vardır. En basit sinir ağı olan percepton, bu kavram ilk kez 1958 yılında Cornell Üniversitesi'nden psikolog Frank Rosenblatt tarafından öne sürüldü. Mantık devrelerinde kullanılan mantık kapılarına benzer. Girdileri direkt olarak çıkış katmanına iletilir ve sonraki katmana eklenebilir. Genel olarak iki girdi ve bir çıkıştan oluşan yapıları vardır.

Tek katmanlı sinir ağları, genellikle sınırları net bir şekilde ayıran ve genellikle iki sınıf içeren basit sınıflandırma problemlerini çözmek için kullanılır. Bu modeller, sınıfları birbirinden ayıran karar doğrusunu belirlemeye odaklanırken, sınırların dışını kestirmek yerine sınıfları net bir şekilde ayırmayı amaçlar. Birden fazla sınıf içeren durumlarda ise, birden çok tek katmanlı sinir ağı kullanılarak her sınıf için ayrı bir karar doğrusu çizilir. Bu iki sınıfı birbirinden ayıran karar doğrusunu (diskriminant), Şekil 2.4.'te gösterildiği gibi çizmeye çalışırlar.



Şekil 2.4. Basit Sınıflandırma Örneği

Varsayıma göre diskirimantlatrın x girdisine bağlı olduğunu düşünürsek diskriminantlar Denklem (2.1)'deki gibi ifade edilebilir. w ağırlıkları, x girişleri simgeler.

$$g(x|w_i, w_{i0}) = w_i^T + w_{i0} = \sum_{j=1}^{d} w_{ij}x_j + w_{i0} \qquad (2.1)$$

Denklem (2.1), çıktının xj girdilerinin ağırlıkları toplamını ifade eder. Burada wj ağırlığı, ilgili girdinin önemini belirtir; dolayısıyla büyük ağırlıklar önemli özellikleri taşır. Genellikle, çıktı, farklı özelliklerin toplamı olarak yazılır.

Denklem (2.1) genelleştirilirse;

$$g(x|W_i, w_i, w_{i0}) = x^T W_i x + w_i x + w_{i0} \qquad (2.2)$$

Kapsamlı modeller oluşturmada 1. Dereceden diskriminantlar yetersiz kalmaktadır. Kapsamlı modeller için yüksek dereceli diskriminant denklemlerine ihtiyaç duyulur. Ancak, bu denklemleri oluşturmak genellikle büyük veri kümelerini gerektirir. Küçük veri kümeleriyle yüksek dereceli diskriminant denklemleri kurulduğunda, aşırı öğrenme riskiyle karşılaşılabilir.

Diskriminantı $\emptyset ij(x)$ olarak tanımlarsak, genel formül

$$g_i(x) = \sum_{j=1}^{k} w_j \emptyset_{ij}(x) \qquad (2.3)$$

olarak ifade edilebilir.

$x1$ ve $x2$ girdilerini z'ler olarak tanımlayarak;

$$z_1 = x_1 \qquad (2.4)$$

$$z_2 = x_2 \qquad (2.5)$$

$$z_3 = x_1^2 \qquad (2.6)$$

$$z_4 = x_2^2 \qquad (2.7)$$

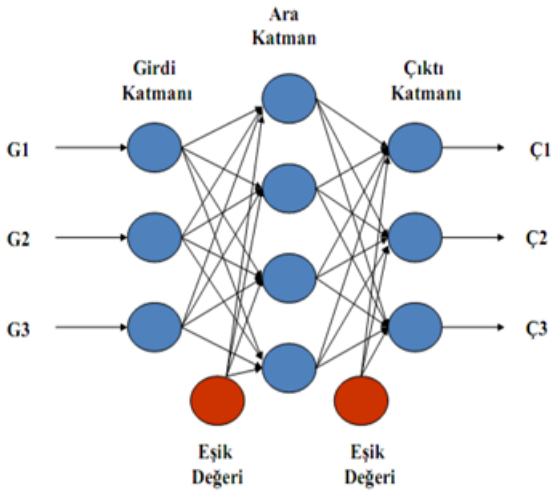
$$z_5 = x_1 x_2 \qquad (2.8)$$

$$z = [z_1, z_2, z_3, z_4, z_5] \qquad (2.9)$$

Denklem (2.9) ile elde edilen z vektörü, 5 boyutlu bir z uzayında ifade edilen diskriminantı, 2 boyutlu x uzayında doğrusal olmayan bir diskriminant olarak tanımlamamıza olanak sağlar. Bu yöntem, doğrusal olmayan bir diskriminant fonksiyonunu doğrudan öğrenmek yerine, veriyi doğrusal olmayan bir dönüşüme tabi tutarak yeni bir uzaya taşır ve orada doğrusal bir fonksiyonla sınıflandırır. Bu yaklaşım, doğrusal olmayan fonksiyonları daha basit bir şekilde ele almamıza imkan tanır.

2.1.4. Çok Katmanlı Sinir Ağı Modelleri

Doğrusal bir şekilde ayrılamayan sınıflandırma ve regresyon problermlerinde tek katmanlı sinir ağı yetersiz kalmaktadır. Çok katmanlı sinir ağı temellerini 1960 yılında Widrow ve Hoff yapmıştır. Çok katmanlı sinir ağı birden fazla algılayıcı katmanlardan oluşmaktadır. Şekil 2.5'de katmanlar şematize edilmiştir.



Şekil 2.5 Çok katmanlı Sinir Ağı Modeli

Çok katmanlı yapay sinir ağları, girdi, gizli ve çıktı katmanlarından oluşur. Bu ağlar, karmaşık problemlerin çözümünde etkilidir ve doğrusal olmayan problemlerde tercih edilir. Birden çok girdi ve gizli katman içerebilirler. Gizli katmanlar, problem yapısına göre esneklik kazandırır ve farklı işlevlerle çıktı katmanına aktarılarak sonuç elde edilir.

Genelde üç katmanlı olarak tanımlanan bu ağlar, eğitim için girdi verilerini işler. Ancak, her katmandaki tüm algılayıcılar beslenir ve çıktıları bir sonraki katmana iletilir. Bu süreç, son katmana ulaşılıncaya kadar devam eder. Özetle, ilk katman basit karar süreçlerini yönetirken, sonraki katmanlar daha karmaşık karar süreçlerini ele alır.

2.1.5. İleri Yönde Yayılım (Forward Propagation)

İleri beslemeli sinir ağlarında, nöronlar girdiden çıktıya kadar düzenli katmanlar halinde dizilirler. Her bir katman kendisini takip eden sonraki katmanlara bağlanır. Giriş katmanından gelen bilgiler doğrudan orta noktaya, yani gizli katmandaki hücrelere iletilir, ardından sırasıyla çıkış katmanından geçerek dış ortama aktarılır. Bu süreçte ileri yönde yayılan bilgi, giriş verilerinin kullanılarak ağın mimarisi boyunca yapılan hesaplama işlemidir.

Her bir nöron, girdi verilerini, kendisine atanmış olan ağırlık ve bias değerlerini kullanarak son katmanda bir tahmin çıktısı üretir. Bu işlemde bias değerleri, her bir nöronun aktivasyon fonksiyonunu değiştirmeye yardımcı olan sayısal değerlerdir. Hesaplamalar, giriş verilerini bir ağırlık değeriyle çarparak ve bir bias değeriyle toplayarak aktivasyon fonksiyonundan geçirir. Aktivasyon fonksiyonları, modelin doğrusallığını kırmak için kullanılır, çünkü gerçek hayattaki sınıflandırma problemleri genellikle doğrusal değildir.

Her bir nöron için hesaplama işlevi aşağıdaki denklemlerle ifade edilir:

$$Z = X * W + b \qquad (2.10)$$

$$\hat{Y} = \sigma(Z) \qquad (2.11)$$

X giriş verilerini, W ağırlıkları, b bias değerini ve σ fonksiyonu aktivasyon fonksiyonunu temsil eder.

2.1.6. Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları, yapay sinir ağı modellerinde doğrusal girdileri, daha karmaşık ve doğrusal olmayan çıktılara dönüştürmek için kullanılırlar. Bu, ağın daha derin ve karmaşık ilişkileri öğrenmesine olanak tanır, böylece daha yüksek dereceli polinomları modelleyebilirler.

Öte yandan, aktivasyon fonksiyonlarının bir diğer önemli özelliği türevlenebilir olmalarıdır. Seçilen aktivasyon fonksiyonları, geri yayılım algoritmasıyla ağı

eğitirken türevlenebilir olmalıdır. Aksi takdirde, ağın ağırlıkları güncellenemez ve model eğitimi gerçekleştirilemez.

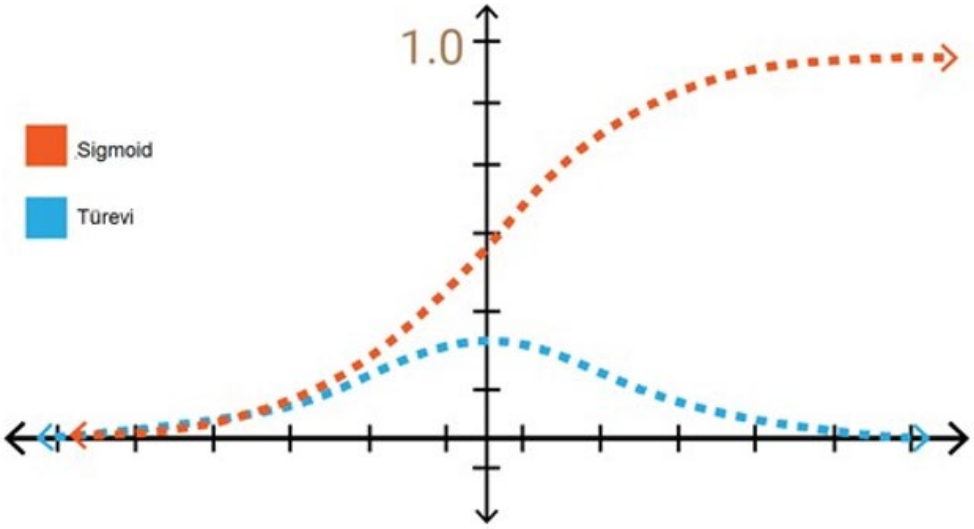
Yapay sinir ağları, gerçek dünya verilerini öğrenmek ve karmaşık yapıları tespit etmek için kullanılan güçlü araçlardır. Ancak, aktivasyon fonksiyonları olmadan, bir yapay sinir ağı verilerdeki karmaşıklıkları anlamakta ve işlemekte zorlanır. Aktivasyon fonksiyonları, ağın her katmanında giriş verilerini dönüştürerek, daha derin ve karmaşık ilişkileri modellemesine olanak tanır.

Aktivasyon fonksiyonlarının olmaması durumunda, yapay sinir ağları sadece basit lineer ilişkileri öğrenebilir, bu da genellikle karmaşık veri yapılarını tespit etmede yetersiz kalır.

Bu nedenle, yapay sinir ağlarının gerçek dünya verilerinden doğru sonuçlar üretmesi ve karmaşık yapıları başarıyla öğrenmesi için aktivasyon fonksiyonlarının kullanılması kaçınılmazdır. Aktivasyon fonksiyonlarının entegrasyonu, yapay sinir ağlarının geniş bir veri yelpazesine uyum sağlamasını ve daha güçlü, daha etkili sonuçlar elde etmesini sağlar. Yapay sinir ağı modellerinde sıklıkla kullanılan aktivasyon fonksiyonları aşağıda yer verilmiştir:

1- **Sigmoid:** Girdiyi 0 ve 1 arasında basit olasılıksal değere dönüştüren S şeklinde olan bir foksiyondur. Sigmoid fonkisyonun denklemi ve grafiği aşağıda belirtilmiştir.

$$\sigma(x) = \frac{1}{1 + e^{-x}} \tag{2.12}$$

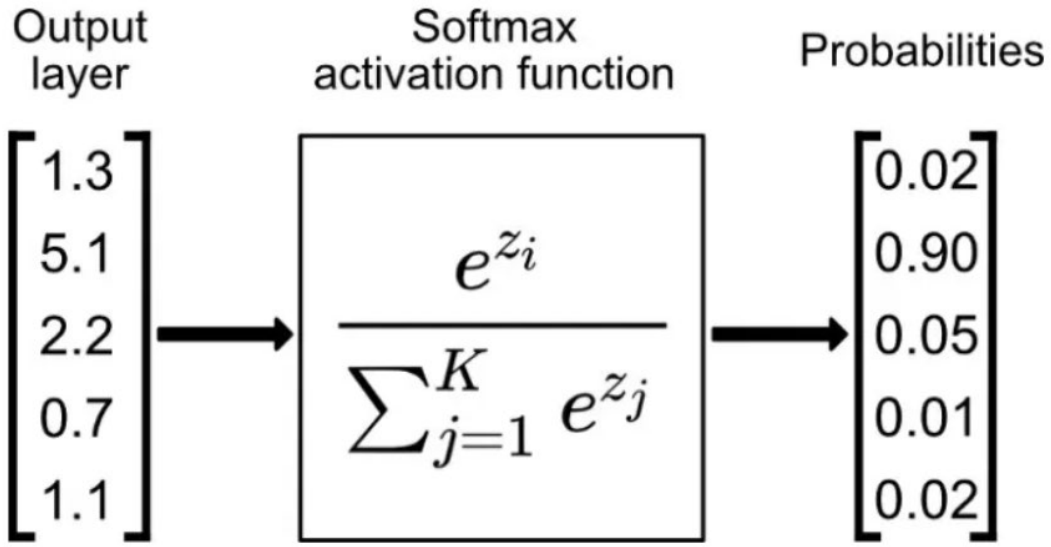


Şekil 2.6. Sigmoid Fonksiyonu ve Türevi

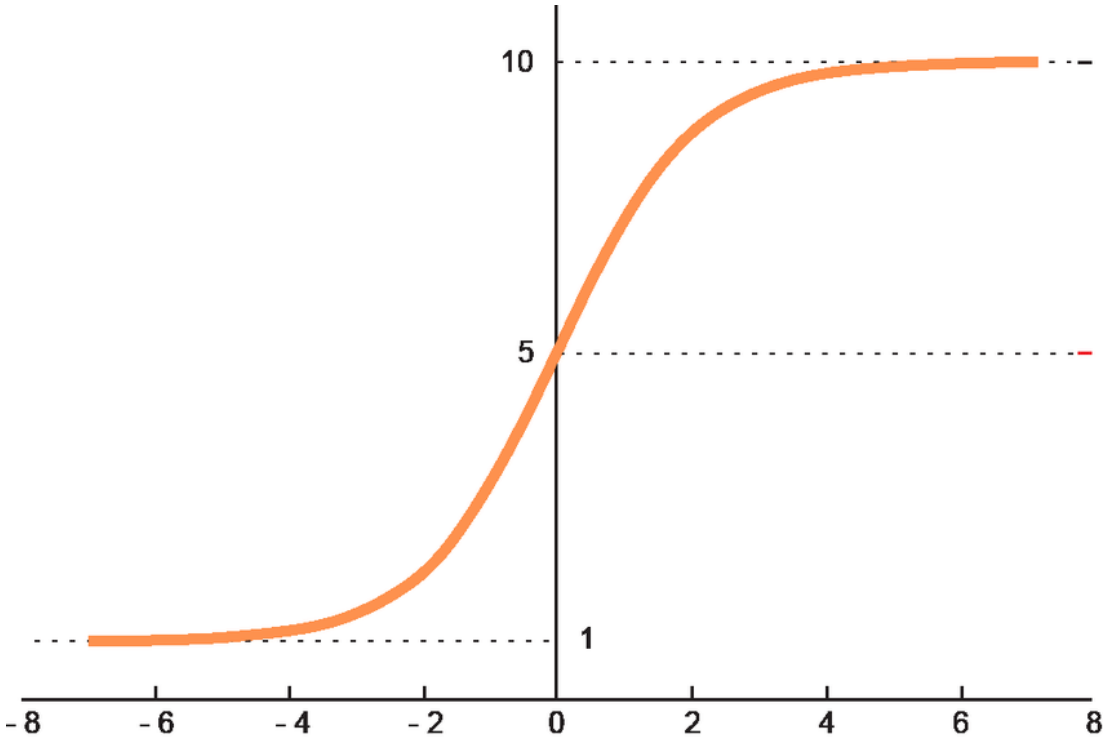
2- **Softmax:** Bir olayın n tane olay üzerindeki olasığını hesaplar bu yönüyle Sigmoid fonksiyonuna benzer. Fonksiyon genellikle sınıflandırma ağlarının çıktı katmanında yer almaktadırlar. Çok sınıflı hedef değişkenleri içeren sınıflandırma problemleri için uygun bir aktivasyon fonksiyonudur. Softmax fonksiyonu ise bu tür problemler için idealdir; çünkü çıktı olarak her bir sınıfa ait olasılık dağılımını döndürür.

Örneğin, Softmax fonksiyonunu kullanarak bir sinir ağı oluşturduğunuzda, çıktı katmanınızda hedef değişkeninizin her bir sınıfı için bir nöron bulunmalıdır. Yani, örneğin 3 sınıflı bir tahmin yapmak istiyorsanız, çıktı olarak [0.2, 0.88, 0.45] gibi her bir sınıfa ait olasılıkları elde edersiniz.
Softmax fonksiyonu Şekil 2.8, örnek Şekil 2.7 ve Denklem 2.13. aşağıda yer verilmiştir.

$$\sigma(x)_i = \frac{e^{x_i}}{\sum_{j=1}^{n} e^{x_j}} \qquad (2.13)$$



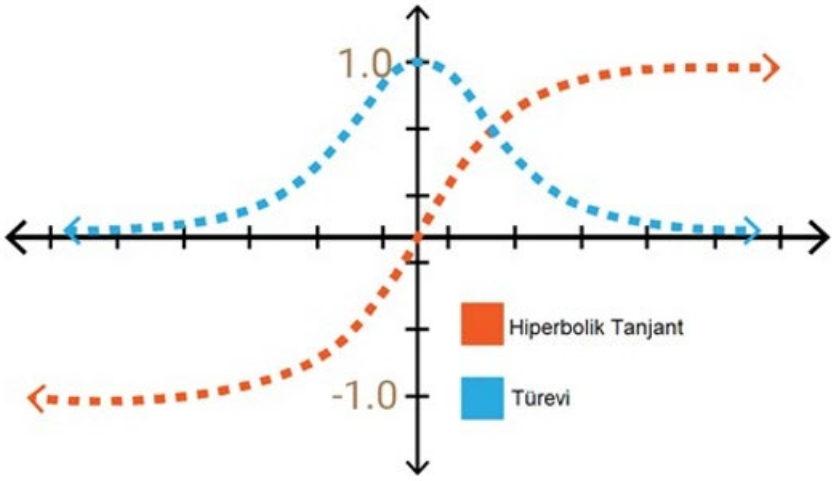
Şekil 2.7 Softmax Fonksiyonu Olasılık Değerleri



Şekil 2.8 Softmax Fonksiyonu ve Türevinin Grafiği

3- **Tanh:** Hiperbolik tanjant olarak da bilinen Tanh fonksiyonu Sigmoid fonksiyonuyla benzer yönü vardır. İkili sınıflandırma için kullanılır. Hiperbolik tanh[-1,1] arasında değer almaktadır ve orjin etrafında simetriktir. Negatif değerlerin daha kolay ele alınabilmesi avantaj sağlamaktadır. Aşağıda Denklem 2.4 ve fonksiyon grafiği Şekil 2.9'da belirtilmiştir.

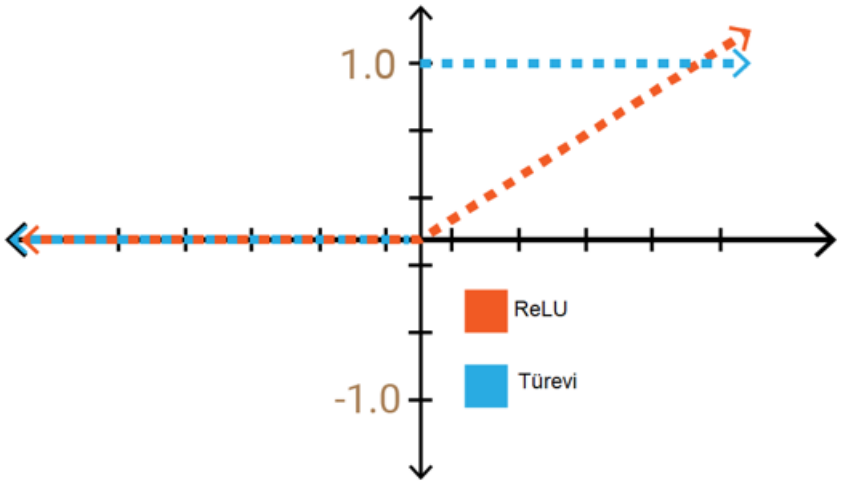
$$\sigma(x) = \frac{\sinh(x)}{\cosh(x)} \tag{2.14}$$



Şekil 2.9: Tanh Fonksiyonu ve Türevinin Grafiği

4- **Rectified Linear Unit (ReLU):** Relu fonksiyonu gizli katmanlar için kullanılmaktadır. ReLu (Rectified Linear Unit) aktivasyon fonksiyonu, bir yapay sinir ağı düğümünün etkinleştirilmesinde temel bir rol oynar. Doğrusal bir işlevden oluşur ve gelen bir değerin pozitif olması durumunda düğümü etkinleştirir, yani çıktı olarak gelen değeri doğrudan alır. Ancak, gelen değer negatif ise çıktı 0 olur. Yani, doğrusal işlevin çıktısı 0'ın üzerindeyse, ReLu fonksiyonu giriş olarak aldığı ham sayıyı çıktı olarak verir; aksi takdirde çıktı 0 olur. ReLU fonksiyonunun Denklem (2.15) ve Şekil 2.10 'da aşağıda belirtilmiştir.

$$\sigma(x) = \max(x, 0) \tag{2.15}$$



Şekil 2.10: ReLU Aktivasyon Fonksiyonunun Grafiği ve Türevi

2.1.7. Loss(Kayıp,Yitim) Fonksiyonun Hesaplanması

İleri yayılma bittikten sonra, eğitim sürecindeki adım olan kayıp fonksiyonu yardımıyla modelin tahmininin gerçek değerlere göre ne kadar iyi veya kötü olduğunu yani hata hesaplaanır. Ulaşılması gereken en ideal değer 0'dır. Hatanın 0 olması gerçek ve tahmin değeri arasında sapma olmadığını ifade etmektedir.

Kayıp fonksiyonu minimize etmek için eğitim sürecindeki ağırlık ve bias değerlerinin güncelleme işlemleri yapılır. Aşağıda en çok kullanılan kayıp fonksiyonlar anlatılmıştır.

1- **Ortalama Karesel Hata (MSE):** Regresyon modelinin performansını ölçmek için kullanılır. Gerçek değer ile tahmin değeri arasındaki farklarının karelerinin toplamını hesaplar.

$$loss = \frac{1}{n}\sum_{i=1}^{n}(y_i - \hat{y})^2 \tag{2.16}$$

n; örneklerin sayısını, y_i; gerçek değerleri, $\hat{y}$ modelin tahmin ettiği değerleri ifade eder.

2- Çapraz Entropi (Cross-entropy): Geleneksel olarak ikili ve çoklu sınıflandırma modelleri için kullanılabilir. İki olasılık dağılımı arasındaki farklılığı ölçer.

$$loss = -\frac{1}{n}\sum_{i=1}^{n} y_i * \log(\hat{y}) + (1 - y_i) * \log(1 - \hat{y}) \tag{2.17}$$

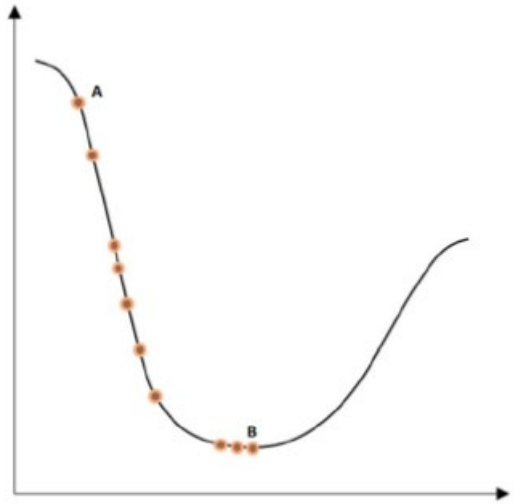
n; örneklerin sayısını, y_i; gerçek değerleri, $\hat{y}$ modelin tahmin ettiği değerleri simgeler.

3-Kategorik Çapraz Entropy (Categorical Cross Entropy): Gerçek sınıf dağılımı ile tahmin edilen sınıf dağılımı arasındaki farkı ölçer. Her bir sınıf için, gerçek sınıf dağılımının logaritmik kaybı alınarak toplam kategorik entropi hesaplanır. Softmax aktivasyon fonksiyonunun çıkışına bağlı olarak kullanılır.

Çapraz entropi ve kategorik entropi arasındaki temel fark, kullanıldıkları senaryolar ve hesaplama yöntemleridir. Çapraz entropi, genellikle iki olasılık dağılımı arasındaki farkı en aza indirmek için kullanılırken, kategorik entropi çok sınıflı sınıflandırma problemlerinde kullanılan ve gerçek ve tahmin edilen sınıf dağılımı arasındaki farkı ölçen bir kayıp fonksiyonudur.

2.1.8. Geri Yönde Yayılım (Backward Propagation)

Eğitim sürecindeki son aşamadır. Ağırlık ve bias değerlerine göre kayıp fonksiyonun kısmi türevlerini (gradyan) hesaplayarak her bir katman için ağırlıkları ve bias değerlerini güncellemek için ağ yapısında sağdan sola doğru ilerleme işlemidir. Ağ çıkışında hesaplanan kayıp fonksiyonunun değeri indirgenmiş olur.

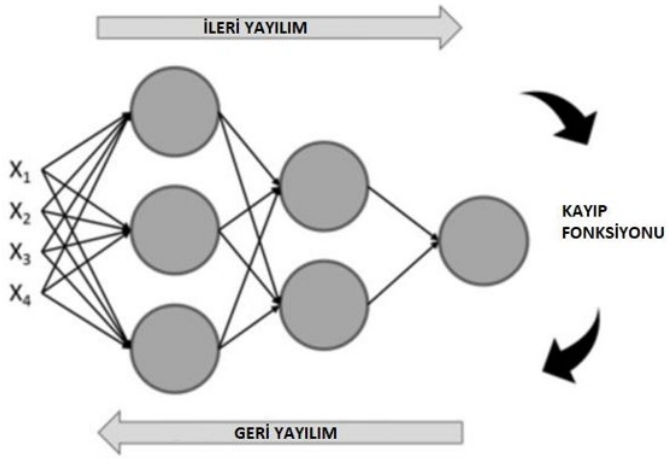


Şekil 2.11: İki boyutlu uzayda yineleme adımlarıyla kayıp fonksiyonu optimizasyonu.

Şekil 2.11 ‘de A noktası herhangi bir optimizasyon öncesi durumu ve kayıp fonksiyonun başlangıç değeridir. Eğrinin altında kalan B noktası Optimizasyon adımlarından ardından birkaç tekrardan sonra kayıp fonksiyonun minimum noktasıdır. Kayıp fonksiyonun en iyi performans gösterdiği ve modelin en iyi tahmin verdiği noktadır.

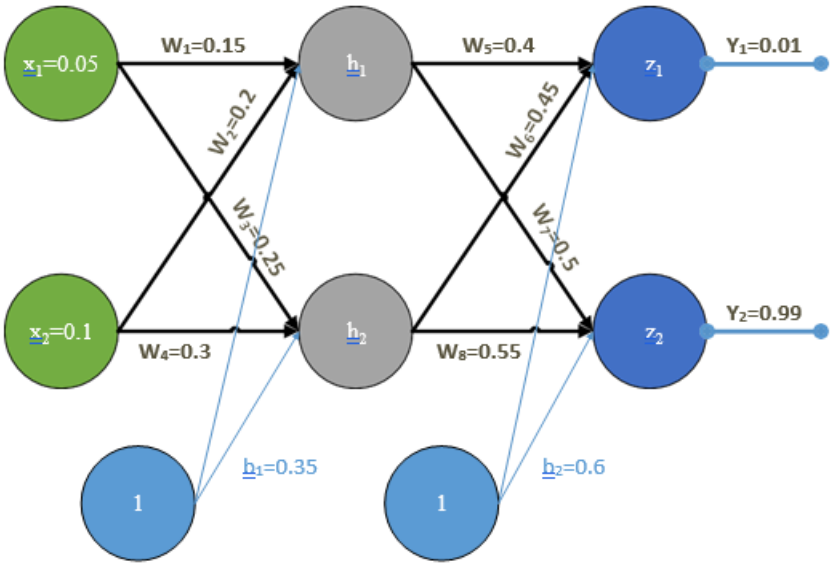
2.1.9. Yapay Sinir Ağlarında Öğrenme Süreci

Yapay sinir ağlarında öğrenme süreci, veri setlerindeki desenleri tanıma ve tahminlerde bulunma yeteneği kazandırmak için ağın ağırlıklarının ve bias'larının ayarlanmasıyla gerçekleştirilir. Bu süreç, genellikle ileri yayılma ve geri yayılma adımlarını içerir.



Şekil 2.12: Yapay Sinir Ağlarının Öğrenme Diyagramı

Yapay sinir ağlarında öğrenme süreci giriş katmanından başlar sırayla gizli katman ve son olarak çıkış katmanında sonlanır. Amaç giriş verileri ile çıkış verileri arasındaki farkı azaltmak. Aşağıda Şekil 2.13'de x = [0.05, 0.1] giriş verileri, y = [0.01, 0.99] çıkış verileri, w = [0.15, 0.2, 0.25, 0.3, 0.35, 0.4, 0.45, 0.5, 0.55] ağırlıklar ve b = [0.35, 0.6] bias değerleridir.

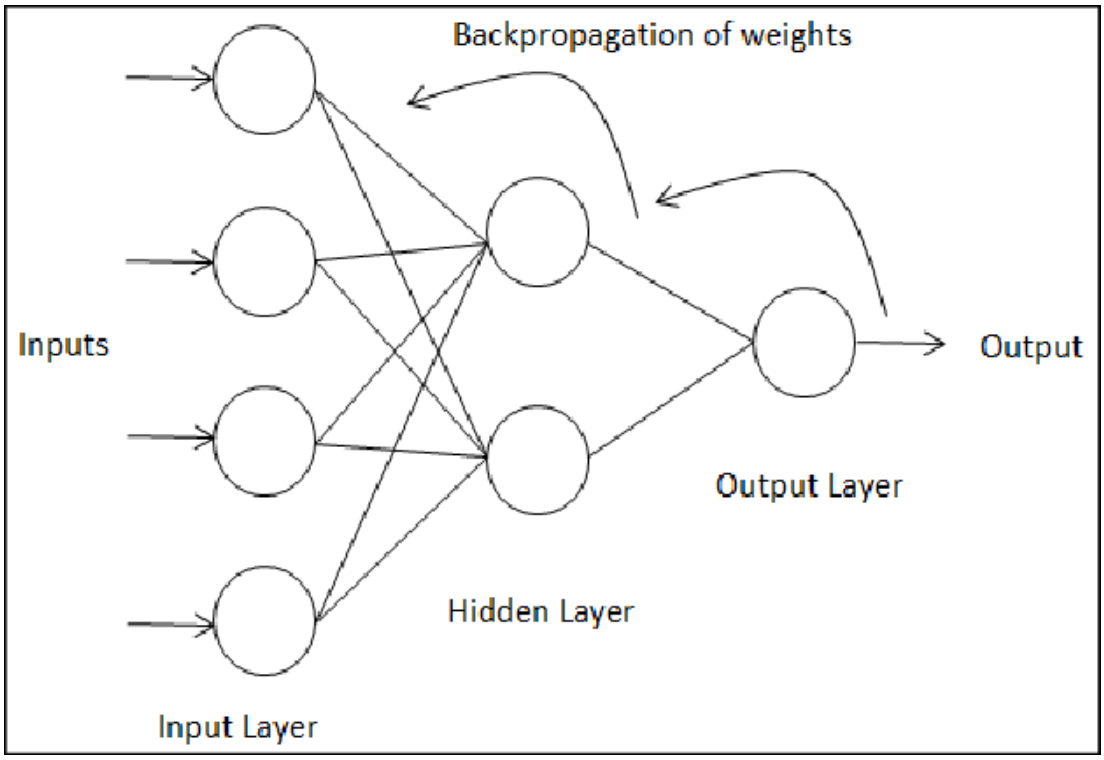


Şekil 2.13: İki Katmanlı Yapay Sinir Ağı Yapısı

İleri yayılma adımı, ağa bir giriş verisi sağlandığında, verilerin ağ boyunca yayılmasını ve her katmandaki nöronlarda hesaplamaların yapılmasını içerir. Giriş

verisi, her bir nöronun ağırlıkları ve bias'ları kullanılarak hesaplanan çıktı değerlerini üretmek için katmanlardan geçer.

Geri yayılma adımı, ağın ürettiği çıktıların gerçek değerlerle karşılaştırılması ve bu karşılaştırmanın hata fonksiyonu ile değerlendirilmesiyle başlar. Ardından, bu hata geriye doğru yayılarak, ağdaki her bir ağırlık ve bias parametresinin hata fonksiyonuna olan katkısı hesaplanır.



Şekil 2.14: Hata Geri Yayılımı

Hata fonksiyonuna göre, ağırlıkların ve bias'ların güncellenmesi, gradyan inişi (gradient descent) yöntemiyle gerçekleştirilir. Gradyan inişi, ağırlık ve bias parametrelerini hata fonksiyonunun minimumuna doğru hareket ettirmek için gradyanın (eğimin) ters yönünde bir adım atar.

Bu süreç, genellikle tüm eğitim verileri üzerinde birden çok kez (epoch) tekrarlanarak gerçekleştirilir. Her bir epoch, ağın daha iyi performans göstermesini sağlamak için ağırlıkların ve bias'ların daha iyi bir şekilde ayarlanmasına yardımcı olur.

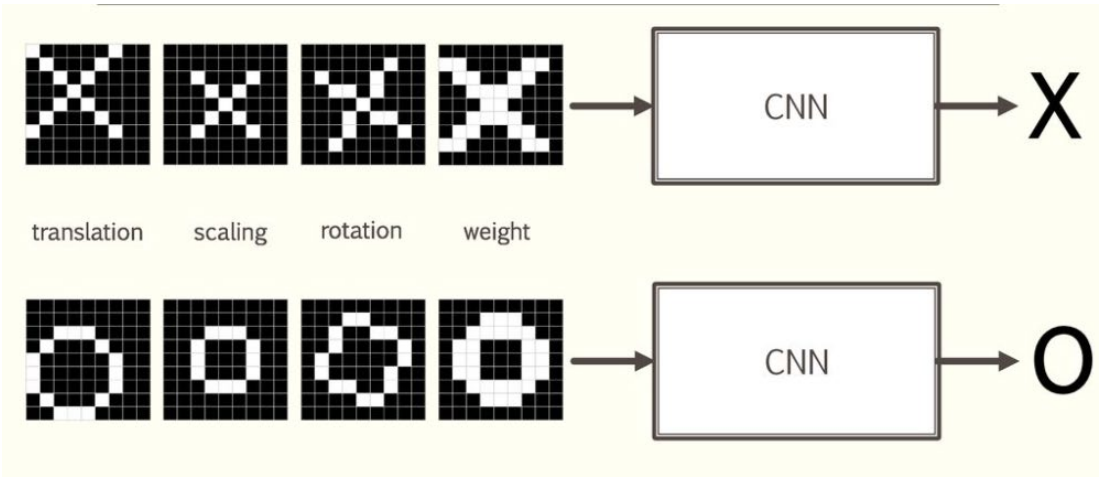
Sonuç olarak, yapay sinir ağlarında öğrenme süreci, ileri yayılma ve geri yayılma adımlarının tekrarlanmasıyla gerçekleştirilir. Bu süreç, ağın eğitilmesi ve verilen görevi başarıyla yerine getirebilmesi için ağın parametrelerinin uygun şekilde ayarlanmasını sağlar.

2.2 Evrişimli Sinir Ağları

Derin öğrenme, son yıllarda büyük bir ilgi görmüş ve birçok alanda çığır açan başarılar elde etmiştir. Özellikle evrişimli sinir ağları (Convolutional Neural Networks), derin öğrenmenin en güçlü araçlarından biri olarak öne çıkmıştır. Ancak, CNN'lerin bazı sınırlamaları da bulunmaktadır.

CNN'lerin en önemli sınırlamalarından biri, girdilerin genellikle görüntüler gibi yapılandırılmış olması gerekliliğidir. Bu, CNN'lerin yalnızca verilerdeki yerel uzamsal kalıpları yakalayabileceği anlamına gelir. Eğer veriler, görüntüler gibi yapılandırılmamışsa, CNN'lerin etkinliği önemli ölçüde azalabilir.

CNN'lerin yapısı, evrişim katmanları ve ardından gelen yoğun katmanlar gibi birbirini takip eden katmanlardan oluşur. Bu katmanlar, girdi verisini özellik vektörlerine dönüştürerek ve ardından sınıflandırma veya başka bir çıktı işlemi için kullanılacak bir sonraki katmana ileterek bilgiyi işler. Her bir katman, öğrenilebilir ağırlıklardan ve bias değerlerinden oluşan nöronlardan oluşur.



Şekil 2.15: CNN ‘in Basit İşlevi

CNN'lerin temel amacı, belirli bir özelliğin bir görüntüde olup olmadığını belirlemek ve bu özelliğin nerede olduğunu kesin olarak belirtmeden sınıflandırmaktır. Bu özelliği, görüntüleri piksel piksel karşılaştırmak yerine, hiper-literal bir şekilde değil, daha genel bir şekilde anlamaya çalışarak gerçekleştirirler.

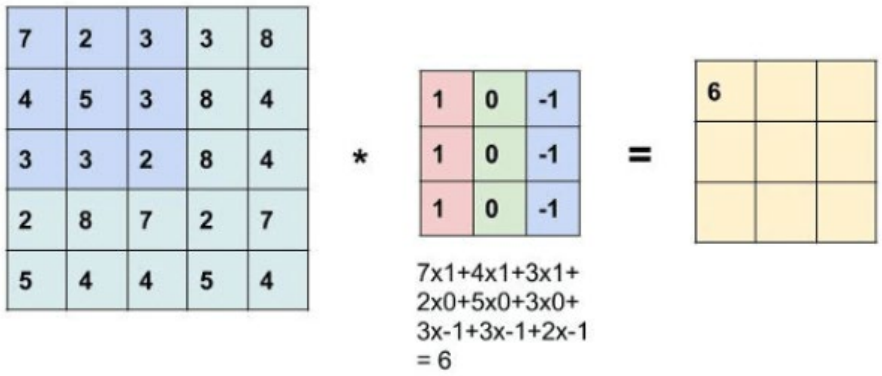
CNN'lerin derin öğrenme yetenekleri, katmanların girdi verisini bir aktivasyon hacmine dönüştürmesi ve bu aktivasyon hacimlerini bir sonraki katmana ileterek bilgiyi işlemesine dayanır. Bu katmanlar, evrişimli ağın ya basit katmanlar dizisi olarak ya da az sayıda karmaşık katman olarak görülmesine bağlı olarak farklı terminolojilerle tanımlanabilir.

2.2.1. Evrişim Katmanı

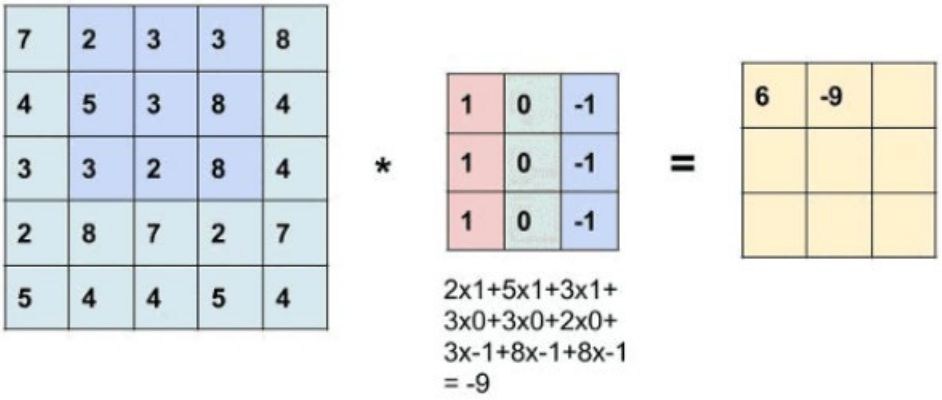
Evrişimli katmanın parametreleri temelde iki veri parçasından oluşur. Girdi ve değerleri öğrenilen bir dizi filtreyle (çekirdekler olarak da adlandırılır) ilgili her şey, yani rastgele verilerle başlar ve eğitim ilerledikçe değiştirilir. [1] Evrişim katmanı, görüntü işleme ve tanıma süreçlerinde kullanılan önemli bir yapılardan biridir. Giriş olarak bir görüntü matrisi ve bu matrise uygulanacak bir filtre matrisi alınır. Filtre matrisi, genellikle küçük boyutlu olup, önceden belirlenmiş öznitelikleri yakalamak için tasarlanmıştır. Evrişim işlemi, bu filtre matrisinin giriş görüntüsü üzerinde gezdirilerek her bir bölge için bir çıktı matrisi oluşturulması işlemidir.

2.2.1.1. Evrişim Aşaması

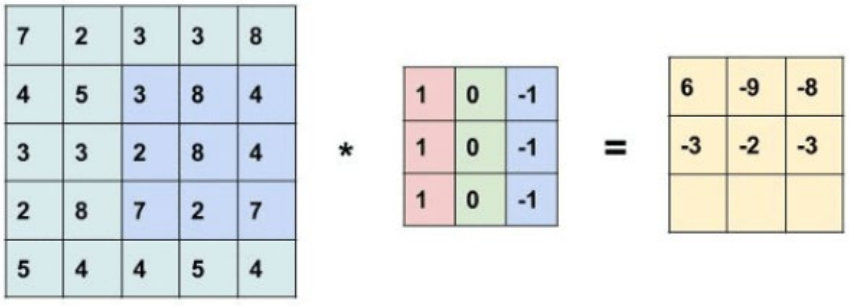
Evrişim işlemi, giriş görüntüsü üzerindeki belli bir bölgeye filtre matrisinin uygulanmasıyla gerçekleştirilir. Bu işlem sırasında filtre matrisi, giriş matrisinin belirli bir bölgesi ile eleman bazında çarpılarak toplanır. Bu toplam sonucu, çıkış matrisinin ilgili konumuna yerleştirilir. Bu işlem, giriş görüntüsü boyunca belirli bir adım (genellikle bir piksel) ile gezdirilerek tekrarlanır.



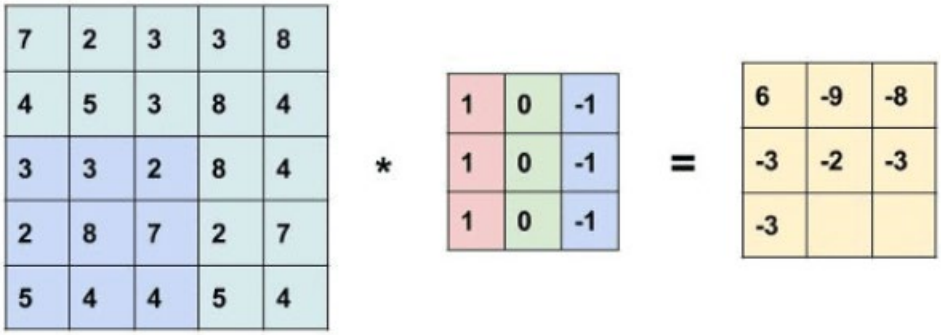
Şekil 2.16: Evrişim İşlemi-1



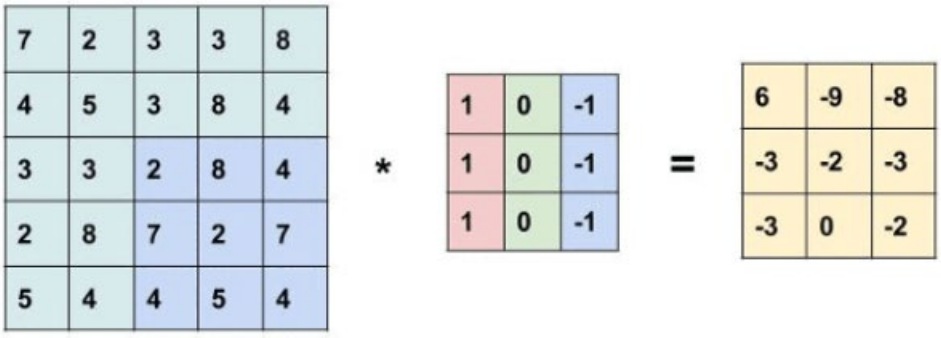
Şekil 2.17: Evrişim İşlemi-2



Şekil 2.18: Evrişim İşlemi-3



Şekil 2.19: Evrişim İşlemi-4



Şekil 2.20: Evrişim İşlemi-5

Şekilde belirtilen 5x5'lik görüntü matrisi ve 3x3 boyutunda filtre matrisi vardır. Bu görüntü üzerinde 3x3'lük filtre matrisi soldan sağa ilerlemektedir ve çıkış matrisi hespalanarak yazılmıştır.

Evrişim işlemi sırasında çıkış matrisinin boyutu, giriş ve filtre matrislerinin boyutlarına bağlıdır. Genel olarak, çıkış matrisinin boyutu şu formülle hesaplanır:

Çıkış Matrisinin Boyutu = Giriş Matrisinin Boyutu – Filtre Boyutu + 1

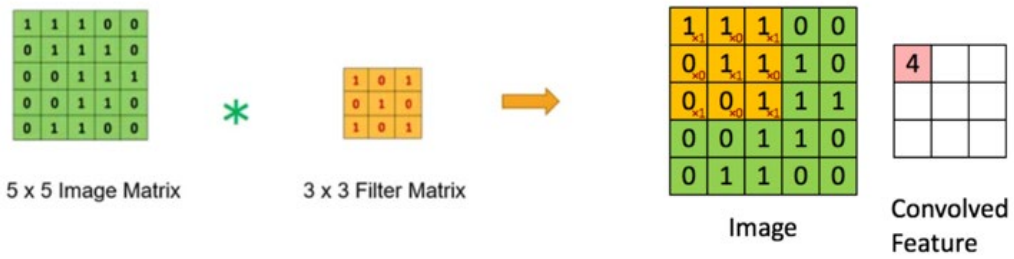
Formül = (n-f+1) x (n-f+1)

Bu formülde, giriş ve filtre matrislerinin boyutları kullanılarak çıkış matrisinin boyutu hesaplanır.

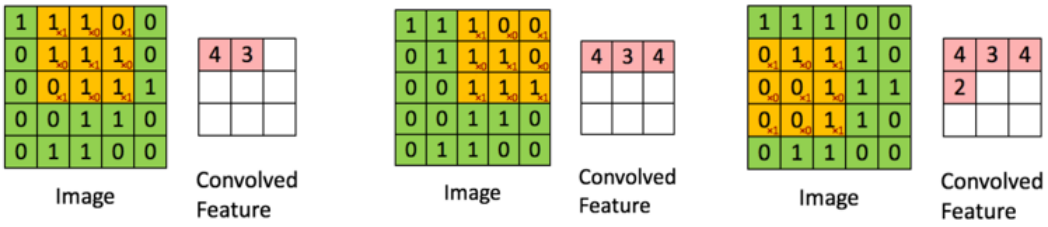
(5-3+1) x (5-3+1) = 3 x 3

Çıkış matrisinin boyutu hesaplandıktan sonra, filtre matrisi görüntü matrisinin sol üst köşesine yerleştirilir. [2] İki matrisin kesişen indislerindeki değerler çarpılarak toplanır, böylece çıkış matrisinin ilk indeksi oluşturulur. Sonra filtre matrisi bir adım sağa kaydırılarak bu işlem tekrarlanır. Filtre matrisi görüntü üzerindeki ilerlemeyi tamamladığında, çıkış matrisindeki değerler hesaplanmış olur. Bu süreç, evrişim katmanında gereken özniteliklerin belirlenmesini sağlar. Ayrıca, birden fazla öznitelik için evrişimli sinir ağlarında birden çok evrişim katmanı kullanılabilir.

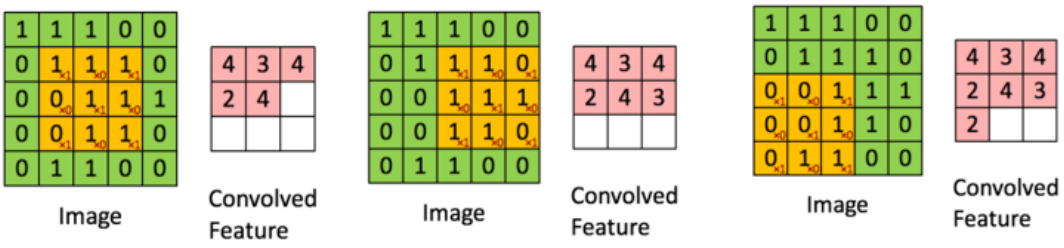
Daha sonra adım kaydırma (Stride) işlemine geçilir. CNN modellerde stride değeri parametre olarak değişebilen değerdir. Stride değeri filtrenin ana görseli üzerinden kaç piksel kayacağını belirleyen işlemdir.



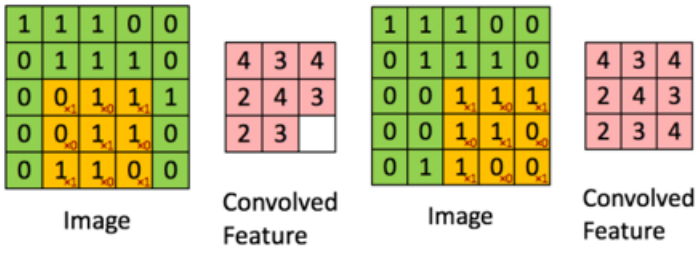
Şekil 2.21: Adım Kaydırma İşlemi (Stride) -1



Şekil 2.22: Adım Kaydırma İşlemi (Stride) -2

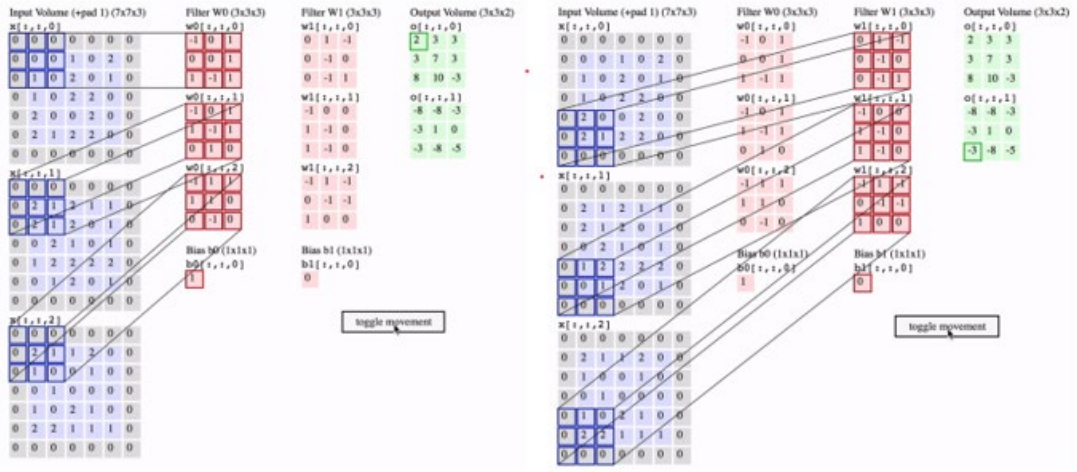


Şekil 2.23: Adım Kaydırma İşlemi (Stride) -3



Şekil 2.24: Adım Kaydırma İşlemi (Stride) -4

Stride değeri 2 olsaydı atlanılan piksel sayısı artacaktı feauture mapte daha küçük hale gelecekti. Aşağıdaki Şekil 2.25'de incelenmiştir.



Şekil 2.25: Stride=2 Olduğu Örnek

Çıkış matrisinin boyutunda değişiklik olmasına sebebiyet vermiştir. Böyle bir durum için çıkış matrisinin boyutunu hesaplarken aşağıdaki Denklem 2.17'de formül ile çözülmektedir.

Çıkış Matrisinin Boyutu=(Giriş Matrisinin Boyutu – Filtre Matrisinin Boyutu) / Stride + 1

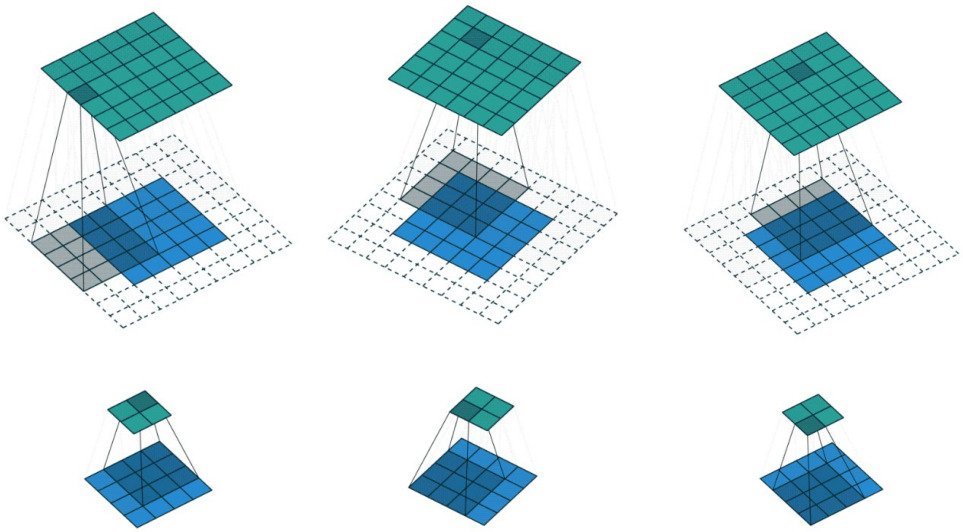
Formül = ((n-f) / S + 1) x((n-f) / S + 1) (2.17)

Şekil 2.25'de stride değeri 2 iken:

Giriş =(nxn)=(7x7), Stride=2

Filtre=(fxf)=(3x3) ama normal şartlarda çıkışımız (5x5) olması gerekir.

Evrişim işleminden sonra giriş ve çıkış matrisi arasında oluşan boyut farkını biçimlendirmek amacıyla "Padding(Piksel ekleme)" işlemi yapılabilir. Giriş matrisi ve çıkış matrisinin aynı boyutlarda olmasını sağlayan bir işlemdir. Padding işlemi, giriş matrisine piksel ekleyerek çıkış matrisiyle aynı boyutta olmasını sağlamaktadır. Aşağıda Şekil 2.26'da padding işlemi gerçekleştirlmiştir.



Şekil 2.26: Giriş Matrisine Padding İşlemi

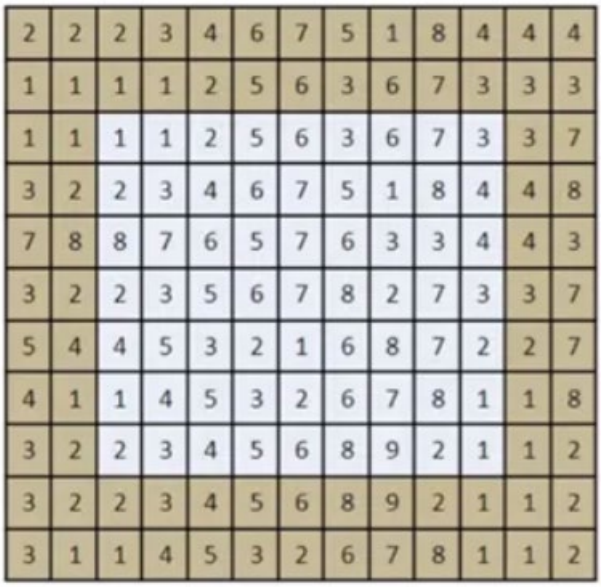
Piksel ekleme işlemi iki yöntem ile yapılabilmektedir:

1- Filtre matrisi görüntü matrisinin üzerinde gezdirilirken görüntü matrisinin dışına taşan kısımlara 0 değeri eklenir. [2]

0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	1	1	2	5	6	3	6	7	3	0	0
0	0	2	3	4	6	7	5	1	8	4	0	0
0	0	8	7	6	5	7	6	3	3	4	0	0
0	0	2	3	5	6	7	8	2	7	3	0	0
0	0	4	5	3	2	1	6	8	7	2	0	0
0	0	1	4	5	3	2	6	7	8	1	0	0
0	0	2	3	4	5	6	8	9	2	1	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0
0	0	0	0	0	0	0	0	0	0	0	0	0

Şekil 2.27 : Piksel Ekleme İşlemi 1. Yöntem

2- Görüntü matrisi dışında kalan kısımlara şekilde görüldüğü üzere komşu indekslerdeki piksel değerleri eklenir. [2]



Şekil 2.28 : Piksel Ekleme İşlemi 2. Yöntem

Şekil 2.28'deki 7x7 boyutunda giriş matrisi vardır. Çıkış matrisi oluşturmak için kullanılacak filtre 5x5 boyutunda olduğunu varsayarsak çıkış matrisinin boyutunu hesaplamak için:

Çıkış Matrisinin Boyutu = (n-f+1) x (n-f+1) (2.18)

= (7-5+1) x (7-5+1)

= 3 x 3

Piksel eklemeden giriş ve çıkış matrisi arasında boyut farkı olduğunu tespit ettik. Piksel ekleme işlemi yaptıktan sonra tekrar çıkış matrisinin boyutunu hesaplaması yapılır.

Padding = (f-1)/2 (2.19)

Formül= (n+2p-f+1) x (n+2p-f+1)

Denklem 2.19'a göre kaç piksel ekleneceği bulunur.

Padding = (5-1) / 2

= 2

Çıkış matrisinin boyutunu tekrardan hesaplaması yapılır.

Çıkış Matris Boyutu = (7+2x2-5+1) x (7+2x2-5+1)

= 7x7

Giriş ve çıkış arasındaki boyut farkı kaldırılmış oldu.

Eğer bir evrişim işlemi sırasında piksel ekleme yöntemi olarak sıfır ile doldurmayı tercih edersek, filtre matrisinin giriş matrisi üzerinde gezinmesiyle çıkış matrisinin piksel değerleri hesaplanırken, sıfır değerlerinin bulunduğu bölgeler çıkış matrisinde bir tür boşluk yaratabilir. Bu durum, elde edilen çıkış matrisinde piksel değerleri arasında belirli bir uzaklık oluşturabilir, ki bu genellikle istenmeyen bir durumdur.

Öte yandan, diğer yöntemde olduğu gibi, ilgili alanlara görüntü matrisi içindeki piksel değerlerini eklersek, bu eklenen değerler görüntüye aitmiş gibi davranır. Bu durum, özellikle çıkış matrisinin hesaplanması sırasında piksel değerleri arasında uyumsuzluk olmadığı için, elde edilen çıkışta daha tutarlı sonuçlar elde edilir. Ancak, bu yöntemin dezavantajı, diğer yönteme kıyasla işlem yükünün artmasıdır.

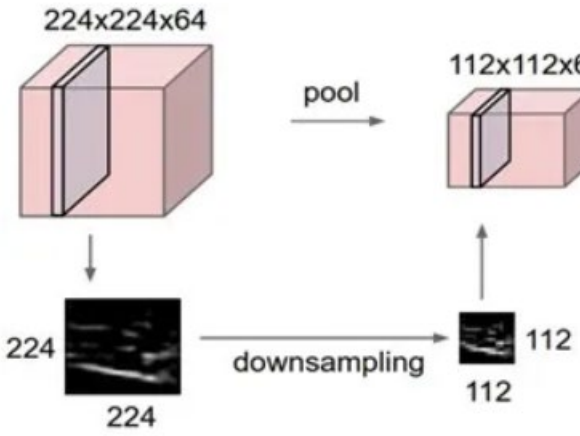
2.2.2. Aktivasyon Katmanı (Activation Layer)

Yapay sinir ağına ses, görüntü, video gibi karmaşık gerçek dünya verilerini öğretmek için aktivasyon fonksiyonuna ihtiyaç duyulur. Aktivasyon fpnksiyonu içermeyen yapay sinir ağında basit lineer regresyon modelinden bir farkı yoktur. Aktivasyon işlemi giriş sinyali üzerinden gerçekleştirildiği için diğer çıkış yaparken hemen ardındaki nörona girdi olarak yani dönüştürülmüş çıktı olarak devam eder. Aktivasyon fonksiyonları doğrusal olmayan dönüşümdür. Relu, Tanh, Sigmoid ve ReLU fonksiyonlarından bahsedilmiştir.

En yaygın kullanılan aktivasyon fonksiyonu olan ReLU, ağın doğrulukta önemli derecede bir fark yaratmadan çok daha hızlı eğitilebildiği için kullanılmaktedır. Eğitimi verimli bir hale getirmektedir. ReLU katmanı, $f(x) = \max(0, x)$ fonksiyonunu ağ girişindeki tüm değerlere uygular. Temel olarak, bu katman basitçe tüm negatif ve sıfır olan tetikleyicileri 0'la değiştirir. Bu katman, ağın doğrusal olmayan özelliklerini artırır. Sıfırdan küçük veya sıfıra eşit değerlerin aktivasyon fonksiyonundaki girişlerin sıfıra eşitlenmesi, yukarıda tartışıldığı gibi, daha değerli kabul edilen dağınık temsiller üreten gizli birimlerde dağılmaya neden olur.[1]

2.2.3. Havuzlama Katmanı (Pooling Layer)

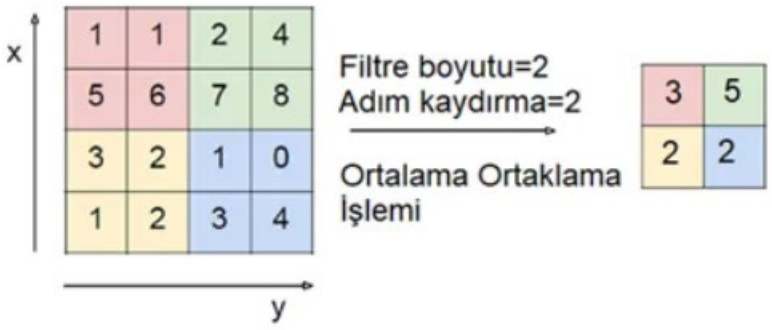
Boyut indirgeme işlemi (Down Sampling) bu katmanda gerçekleşmektedir. Öğrenme işlemi gerçekleşmez, giriş matrisinin kanal sayısı sabit tutulmasını sağlayarak genişlik ve yükseklik bazında boyut indirgeme gerçekleşmektedir. Evrişim katmanından sonra kullanılır çünkü hesaplama karmaşıklığının önüne geçer. Şekil 2.29'da Down Sampling uygulaması gerçekleşmiştir. Çözünürlük azaltılır.



Şekil 2.29: Down Sampling İşlemi

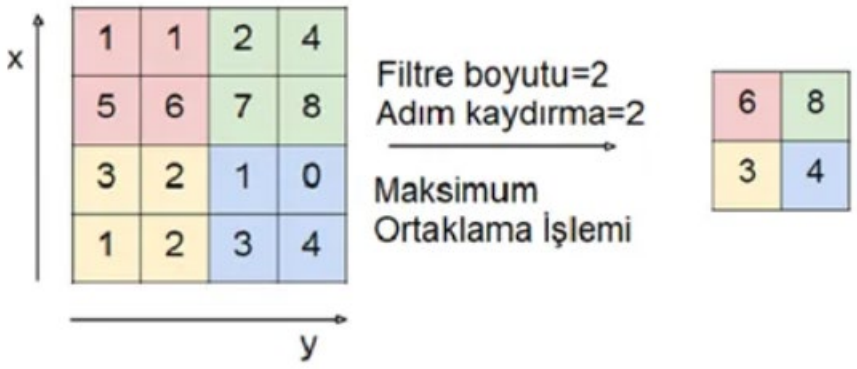
İki yöntemle bu işlem gerçekleşmektedir:

-1 Ortalama Havuzlama (Average Pooling): Filtre, görüntü matrisi üzerinde gezinirken kesişen noktaların piksel değerlerinin ortalamasını alarak boyutu indirger böylece yeni görüntü matrisi oluşmaktadır. Aşağıdaki Şekil 2.30'daki örnek görüntüde 4x4 boyutunda matrise average pooling işlemi uygulandığında 2x2 boyutunda yeni görüntünün matrisi oluşmuş olur.



Şekil 2.30: Ortalama Hazuzlama İşlem Örneği

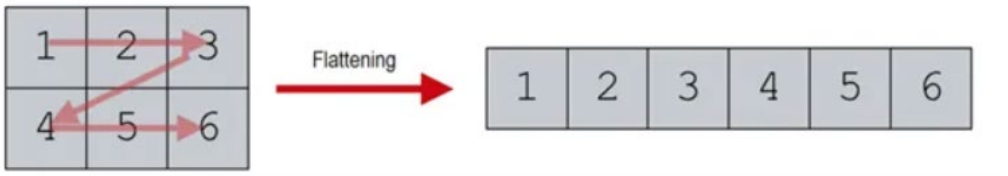
-2 Maksimum Havuzlama (Maximum Pooling): Maksimum havuzlama kullanılırsa filtre görüntü matrisi üzerinde gezinirken kesiştiği noktalarda piksel değerleri arasında maksimum olan değer seçilir böylece boyutu azaltılmış yeni görüntü matrisi elde edilir. Şekil 2.31'deki görüntüde 4x4 boyutlu matrise maximum pooling işlemi uygulandığında 2x2 boyutunda yeni görüntü matrisi oluştu.



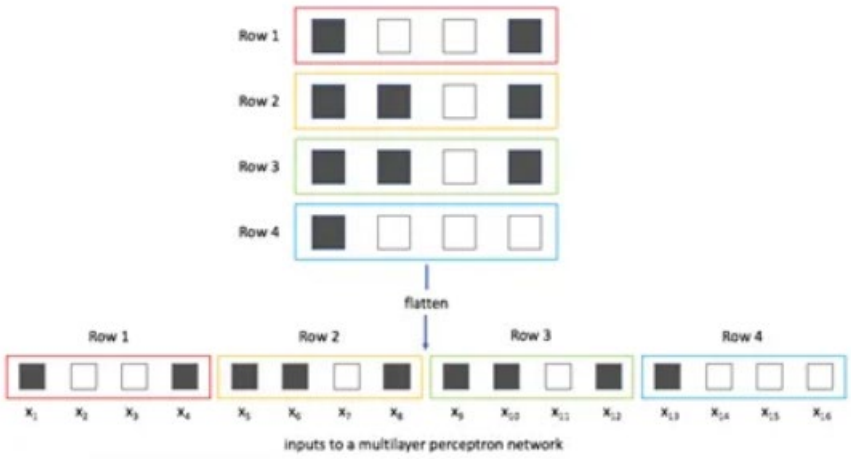
Şekil 2.31: Maksimum Hazuzlama İşlem Örneği

2.2.4. Flattening Katmanı

Tüm işlemler matrisler üzerinden gerçekleşti bu katmanda bir sonraki katmana aktarım yapabilmek için yapay sinir ağlarının istediği formata tek düzlemli vektöre dönüştürür. Full Connected Layer'a giriş verileri hazırlar. Convulational ve Pooling katmanından gelen matrisleri tek düzlem haline getiren katmandır. Şekil 2.32'de Flattening işlemi gerçekleşmektedir.



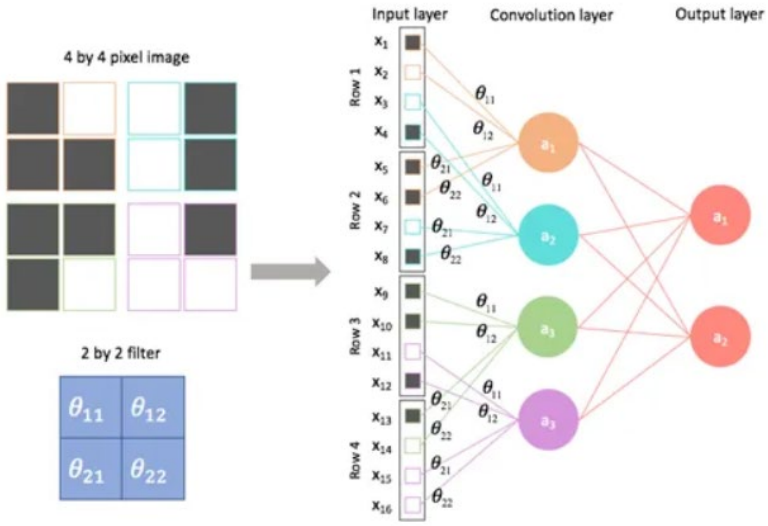
Şekil 2.32: Flettening İşlemi



Şekil 2.33: Flattening Katmanı

2.2.5. Fully Connected Layer (Tam Bağlantılı Katman)

Flattening'den gelen tek düzlemli vektörleri alıp yapay sinir ağlarına giriş olarak verir. Öğrenme işlemi başlatılmış olur.



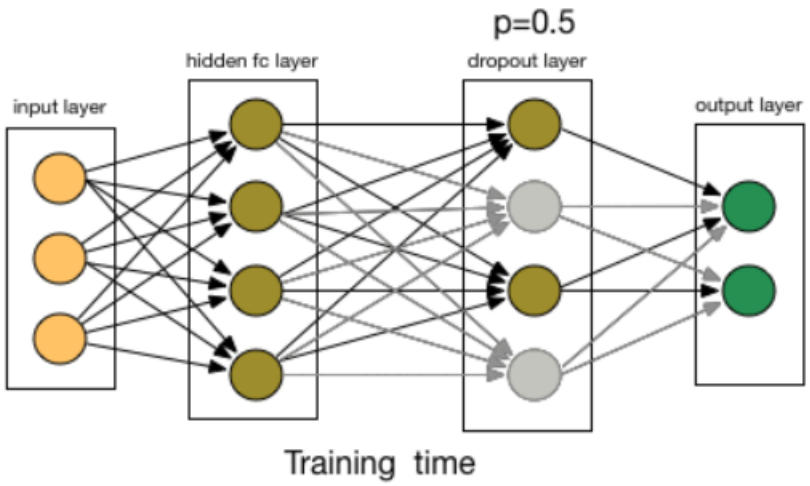
Şekil 2.34: Full Connected Katman

2.2.6. Batch Normalization Layer (Toplu Normalleştirme Katman)

Normalleştirme, verileri standartlaştırmak amacıyla verilere uygylanan önişleme işlemidir. Batch Normalization işlemi sinir ağı katmanları arasındaki yapılan normalleştirme işlemidir. [2] Eğitimi hızlandırmak ve daha yüksek öğrenme değerleri elde etmek için kullanılır. Evrişim ve Aktivasyon katmanı arasında uygulanır.

2.2.7. Dropout Layer (Seyreltme Katmanı)

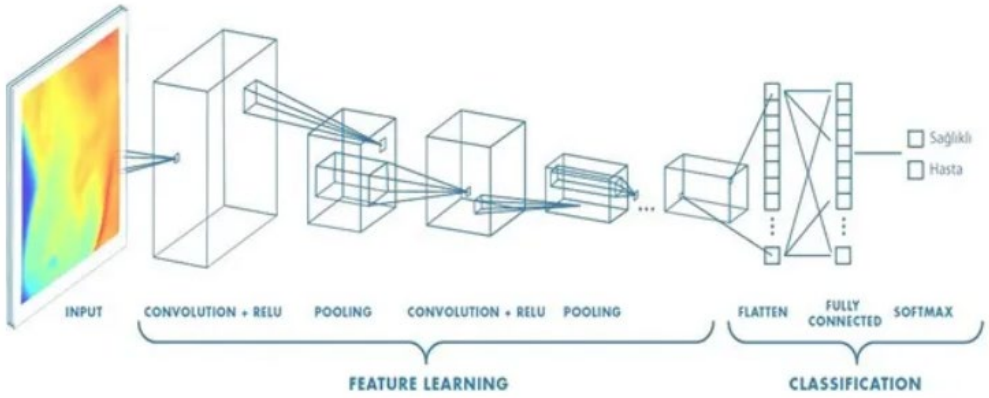
Rastgele seçilen nöron setlerinin eğitimi sırasında nöronların yok sayılmasıdır. Overfitting durumlarında kullanılabilir.



Şekil 2.35: Droput Katmanı

2.2.8. CNN Mimari Yapısı

Bait bir CNN yapısı birkaç Evrişim ve Havuzlama katmanını arka arkaya eklemek, her bir katmandan sonra ReLU katmanı eklemek, Pooling katmanı Flattening katmanı eklenmelidir. CNN'nin en son katmanı Fully Connected katmanıdır.



Şekil 2.36: CNN Mimari Yapısı

2.2.9. Eğitim ve Doğrulama

Ağ, eğitim verilerini işlemek için gizli katmanlardaki ağırlıkları, biasları ve aktivasyon fonksiyonlarını kullanır. Eğitim süreci boyunca, ağ tarafından üretilen çıktılar, beklenen çıktılarla karşılaştırılır ve aralarındaki fark hesaplanır. Bu hatalar, geri yayılım (backpropagation) yöntemi ile ağın katmanlarına dağıtılır. Böylece, her eğitim adımında ağırlıklar ve biaslar düzeltilir. Bu süreç, eğitim verileri üzerinde defalarca tekrarlanır ve her döngüde ağırlıklar güncellenir.

2.2.10. Adam Optimizasyonu

Adam optimizasyon algoritması, derin sinir ağlarının eğitiminde kullanılan bir öğrenme hızı optimizasyon yöntemidir. Bu algoritma, özellikle büyük veri setleri üzerinde eğitim yaparken etkili olan adaptif bir öğrenme hızı mekanizmasına sahiptir. Parametrelerin öğrenme hızı, her bir parametre için ayrı ayrı ve zamanla değişen bir şekilde ayarlanır. Bu sayede, eğitim sırasında performansı artırmak için gereksiz yön değişikliklerini azaltır.Bu algoritma, öğrenme hızını parametrelere göre ayarlar ve seyrek parametreler için daha büyük, sık parametreler için ise daha küçük güncellemeler yapar. Bu nedenle, doğal dil işleme veya görüntü tanıma gibi seyrek verilerle çalışan uygulamalar için oldukça uygundur. Ayrıca, Adam'ın bir diğer avantajı, her parametre için ayrı bir öğrenme hızına sahip olması nedeniyle öğrenme hızının ayarlanmasını basitleştirmesidir.

2.2.11. Softmax Fonksiyonu

Bu aktivasyon fonksiyonu, sinir ağlarının son katmanında sıkça kullanılan bir türdür. Özellikle sınıflandırma problemlerinde, ağın çıkışını olasılık dağılımına dönüştürmek için tercih edilir. Bu fonksiyon, her bir nöronun çıktısını [0,1] aralığına sıkıştırarak bir tür normalleştirme sağlar. Bu sayede, giriş verisinin hangi sınıfa ait olma olasılığını belirlemek daha kolay hale gelir.

Matematiksel olarak, bu fonksiyon giriş verisinin ağırlıklı toplamını alır ve bu toplamı softmax fonksiyonuna sokar. Softmax fonksiyonu, girişlerin bir olasılık dağılımına dönüştürülmesini sağlar, böylece her sınıf için bir olasılık değeri elde edilir. Bu sayede, ağın çıkışı sınıflar arasında bir olasılık dağılımı sunar.

Bu fonksiyonun kullanılması, sinir ağlarının eğitiminde önemlidir çünkü çıkışın olasılık dağılımı elde edilir. Bu da çapraz doğrulama ve entropi kaybı gibi metriklerin hesaplanmasını mümkün kılar. Sonuç olarak, sinir ağlarının doğruluğunu ve performansını değerlendirmek için bu fonksiyon sıkça tercih edilir.

2.2.12. ResNET50 ve Mimari Yapısı

Türkçe karşılığı olarak artık ağlar da denilmektedir. ResNet 2015 yılında Kaiming He, Xiangyu Zhang, Shaoqing Ren ve Jian Sun tarafından geliştirilmiş olup "Deep Residual Learning for Image Recognition" makalesinde tanıtılan belirli bir sinir ağı türüdür.

Derin sinir ağları, karmaşık görevlerde yüksek başarı oranlarına ulaşabilmesine rağmen, ağ derinliği arttıkça öğrenme süreci çeşitli zorluklarla karşı karşıya kalır. Bu zorluklardan en belirgin olanları vanishing gradient ve exploding gradient problemleridir. Bu problemlerin üstesinden gelmek amacıyla geliştirilmiş olan Residual Networks (ResNet), skip connections (kısayol bağlantıları) kullanarak ağın daha derinlemesine öğrenme kapasitesini artırır. Bu tezde, ResNet mimarisi, özellikle ResNet50 ve onun temel yapı taşı olan artık bloklar (residual blocks) detaylı olarak incelenecektir. ResNet, derin sinir ağlarının öğrenme kabiliyetini artırmak ve derin ağların eğitim sürecini iyileştirmek amacıyla geliştirilmiştir. ResNet'in temel yeniliği, kısayol bağlantıları (skip connections) kullanarak, ağı daha derin katmanlarla donatmak ve böylece öğrenmeyi kolaylaştırmaktır.

Derin sinir ağları, karmaşık görevlerde yüksek başarı oranlarına ulaşabilmesine rağmen, ağ derinliği arttıkça öğrenme süreci çeşitli zorluklarla karşı karşıya kalır. Bu zorluklardan en belirgin olanları vanishing gradient ve exploding gradient problemleridir. Bu problemlerin üstesinden gelmek amacıyla geliştirilmiş olan Residual Networks (ResNet), skip connections (kısayol bağlantıları) kullanarak ağın daha derinlemesine öğrenme kapasitesini artırır. Bu tezde, ResNet mimarisi, özellikle ResNet50 ve onun temel yapı taşı olan artık bloklar (residual blocks) detaylı olarak incelenecektir.

ResNet, derin sinir ağlarının öğrenme kabiliyetini artırmak ve derin ağların eğitim sürecini iyileştirmek amacıyla geliştirilmiştir. ResNet'in temel yeniliği,

kısayol bağlantıları (skip connections) kullanarak, ağı daha derin katmanlarla donatmak ve böylece öğrenmeyi kolaylaştırmaktır.

Residual block, ResNet'in temel yapı taşıdır. Bu bloklar, girdiyi doğrudan çıktıya bağlayan kısayol bağlantılarını içerir. Bu yapının matematiksel ifadesi şu şekildedir:

$H(x)=F(x)+xH(x)=F(x)+x$

Burada:

1- xx: Blok girişi

2- $F(x)F(x)$: Öğrenilen fonksiyon (evrişimsel katmanlar, aktivasyon fonksiyonları vb.)

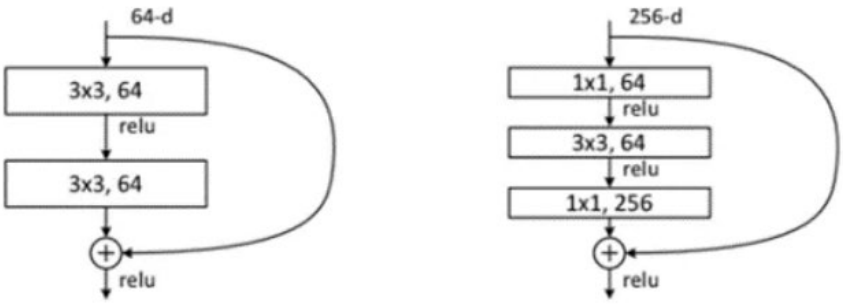
3- $H(x)H(x)$: Blok çıkışı

Bu yapıda, xx doğrudan çıktıya eklenir ve bu sayede öğrenme süreci hızlanır ve stabilize olur.

Bir residual artık blokları genellikle şu bileşenlerden oluşur:

1- İki veya üç evrişimsel katman (Conv2D)
2- Batch normalization katmanları (BatchNormalization)
3- ReLU aktivasyon fonksiyonları
4- Skip connection (kısayol bağlantısı)

Her iki katmanlı bloğu, üç katmanlı darboğaz bloğu ile değiştirerek ResNet 50 ve 34'teki yapısal değişiklikler elde edilir. Bu değişimde her bir artık fonksiyon F için üç katmanlı bir yığın kullanılır. Bu üç katman, sırasıyla 1×1, 3×3 ve yine 1×1 boyutlarına sahiptir. 1×1 katmanı, boyutları azaltıp ardından artırmakla görevlidirken, 3×3 katmanı daha küçük girdi/çıktı boyutlarına sahip bir darboğaz oluşturur. [4]



Şekil 2.37:Sol Resnet-34 İçin Yapıtaşı Sağ Resnet-50/101/152 İçin Darboğaz Yapıtaşı

ResNet50, toplamda 50 katmandan oluşan bir ResNet mimarisidir. Bu mimaride 48 evrişimsel katman, 1 MaxPooling katmanı ve 1 Global Average Pooling katmanı bulunur. ResNet50, derin ağların öğrenme sürecini iyileştirmek amacıyla residual block'ları kullanır.

ResNet50'nin genel yapısı şu şekildedir:

1- İlk Katman: 7x7 evrişim filtresi ve 64 filtre kullanarak evrişim katmanı (Conv2D), ardından bir MaxPooling katmanı.

2- Dört Evrişim Bloğu:

2.1- İlk blok: 3 adet residual block, her biri 64 filtreli.

2.2- İkinci blok: 4 adet residual block, her biri 128 filtreli.

2.3- Üçüncü blok: 6 adet residual block, her biri 256 filtreli.

2.4- Dördüncü blok: 3 adet residual block, her biri 512 filtreli.

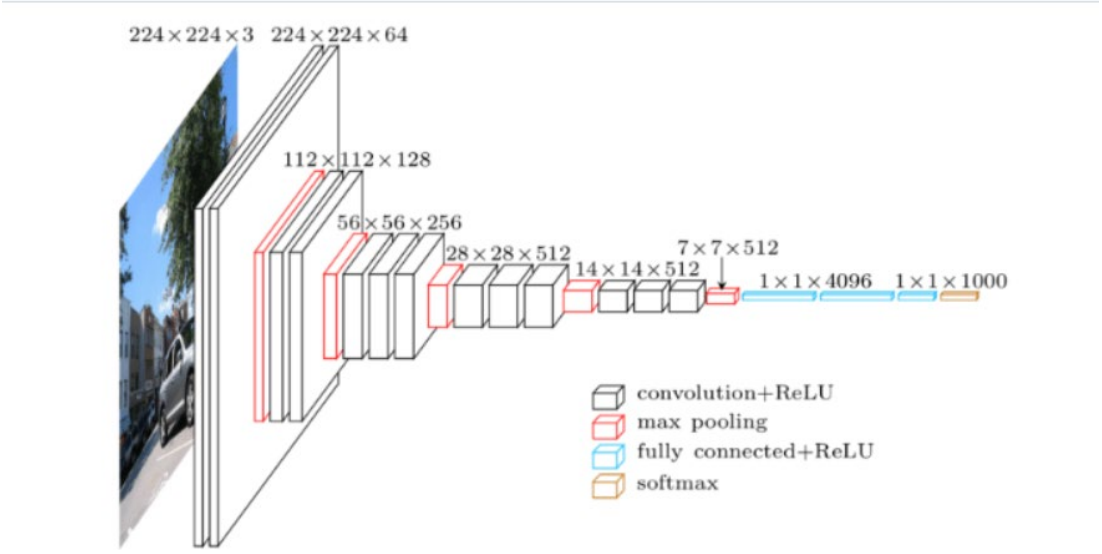
3- Son Katman: Global Average Pooling ve Dense (tam bağlantılı) katman, sınıflandırma için.

ResNet mimarisi, derin sinir ağlarının öğrenme ve performansını artıran yenilikçi bir yapıdır. ResNet50, residual block'lar kullanarak daha derin ağların eğitimini

kolaylaştırır ve vanishing/exploding gradient problemlerinin üstesinden gelir. Bu sayede, daha karmaşık ve derin ağlar bile etkili bir şekilde eğitilebilir ve yüksek performans elde edilebilir. ResNet mimarisinin bu özellikleri, derin öğrenme alanında önemli bir ilerlemeyi temsil eder ve birçok uygulamada başarılı sonuçlar elde edilmesini sağlar.

2.2.13. VGGNet16 ve Mimari Yapısı

Görüntü tanıma ve sınıflandırma gibi bilgisayarla görme görevlerinde derin sinir ağları, yüksek doğruluk oranlarıyla etkileyici sonuçlar elde etmektedir. Bu bağlamda, VGG (Visual Geometry Group) mimarisi, derin öğrenme alanında önemli bir yer edinmiştir. VGG ağları, özellikle VGG16, derinlik ve performans açısından standart hale gelmiş ve çeşitli görüntü işleme görevlerinde yaygın olarak kullanılmıştır.



Şekil 2.38: VGG16 Mimari Yapı

VGG mimarisi, 2014 yılında Simonyan ve Zisserman tarafından önerilmiştir ve temel olarak ardışık evrişim katmanları, max pooling katmanları ve tam bağlantılı katmanlardan oluşur. VGG'nin ana yeniliği, küçük 3x3 filtreler kullanarak daha derin ve daha etkili ağlar oluşturmaktır. VGG ağları, genellikle 3x3 boyutunda filtreler kullanarak arka arkaya evrişim katmanları uygular. Bu katmanlar, non-lineerlik eklemek için ReLU aktivasyon

fonksiyonunu kullanır. Evrişim katmanlarını belirli bir derinliğe ulaştıktan sonra, max pooling katmanları kullanarak uzamsal boyutlar küçültülür. Son olarak, birkaç tam bağlantılı (fully connected) katman ve softmax aktivasyon fonksiyonu ile sınıflandırma yapılır.

VGG16, VGG ağlarının en popüler versiyonlarından biridir. Adındaki "16" sayısı, modelin 16 ağırlık katmanına sahip olduğunu belirtir. VGG16'nın katmanları şu şekildedir:

1- Giriş Katmanı: 224x224x3 boyutunda bir giriş görüntüsü alır.

2- Evrişim Bloğu 1:

    2.1- 2 adet 3x3 evrişim katmanı, 64 filtre
    2.2- 1 adet max pooling katmanı

3- Evrişim Bloğu 2:
    3.1- 2 adet 3x3 evrişim katmanı, 128 filtre
    3.2- 1 adet max pooling katmanı

4- Evrişim Bloğu 3:
    4.1- 3 adet 3x3 evrişim katmanı, 256 filtre
    4.2- 1 adet max pooling katmanı

5- Evrişim Bloğu 4:
    5.1- 3 adet 3x3 evrişim katmanı, 512 filtre
    5.2- 1 adet max pooling katmanı

6- Evrişim Bloğu 5:
    6.1- 3 adet 3x3 evrişim katmanı, 512 filtre
    6.2- 1 adet max pooling katmanı

7- Tam Bağlantılı Katmanlar:
    7.1- 2 adet 4096 nöronlu tam bağlantılı katman
    7.2- 1 adet 1000 nöronlu tam bağlantılı katman (ImageNet veri seti için)

    7.3- Softmax aktivasyon fonksiyonu

VGG16, ImageNet veri seti üzerinde eğitilmiş olup, milyonlarca görüntü üzerinde test edilmiş ve yüksek doğruluk oranlarına ulaşmıştır. VGG16'nın başarısı, mimarisinin derinliği ve küçük filtre boyutlarının etkinliğinden kaynaklanmaktadır. Derin ağ yapısı, daha karmaşık özelliklerin öğrenilmesine olanak tanırken, küçük filtre boyutları daha detaylı özelliklerin yakalanmasını sağlar.

2.2.14. Transfer Learning

Makine öğrenimi ve derin öğrenme, birçok endüstriyel ve akademik alanda devrim niteliğinde etkiler yaratmaya devam ediyor. Yüz tanıma, otomatik çeviri, tıbbi teşhis gibi birçok alanda bu teknolojiler önemli bir rol oynuyor. Ancak, bu karmaşık modellerin sıfırdan eğitilmesi oldukça zaman alıcı ve maliyetlidir. İşte bu noktada Transfer Learning devreye giriyor. Transfer Learning, genellikle geniş ve kapsamlı veri setleri üzerinde önceden eğitilmiş modellerin öğrendiği bilgilerin, yeni ve görece daha küçük veri setlerinde etkin bir şekilde kullanılmasını sağlar. Genellikle, Transfer Learning'in ilk adımı, geniş veri setleri üzerinde eğitilmiş bir modelin alınmasıdır. Bu modeller, özellikle görüntü tanıma, dil işleme gibi belirli görevler için oldukça yetenekli olabilirler. Örneğin, VGG, ResNet gibi modeller milyonlarca görüntü üzerinde eğitilmiştir

3. MATERYAL VE YÖNTEM

3.1 Trafik İşaretlerinin Tarihçesi

Trafik işaretleri, karayollarında bilgi verme, yasaklama ve tehlike belirtme gibi üç ana amaç için kullanılırlar. Ancak, araç teknolojisinin gelişmesiyle birlikte, trafik işaretlerine ve çeşitliliğine olan ihtiyaç artmıştır.

Ulusal düzeyde standart trafik işaretleri kullanımı genellikle yerel yasalara ve kültürel farklılıklara göre değişiklik gösterebilir. Bu durum, uluslararası sürüş deneyimini zorlaştırabilir ve trafik güvenliğini etkileyebilir. Bu nedenle, farklı ülkeler arasında trafik işaretlerinin standartlaştırılması için çaba harcanmıştır.

Bu amaca ulaşmak için 1931 yılında "Geneva Convention Concerning the Unification of Road Signals" adlı bir konferans düzenlenmiştir. Ardından, 1949'da bu konferansın temel alındığı "Geneva Protocol on Road Signs and Signals" konferansı gerçekleştirilmiştir. Son olarak, 1968'de "Vienna Convention

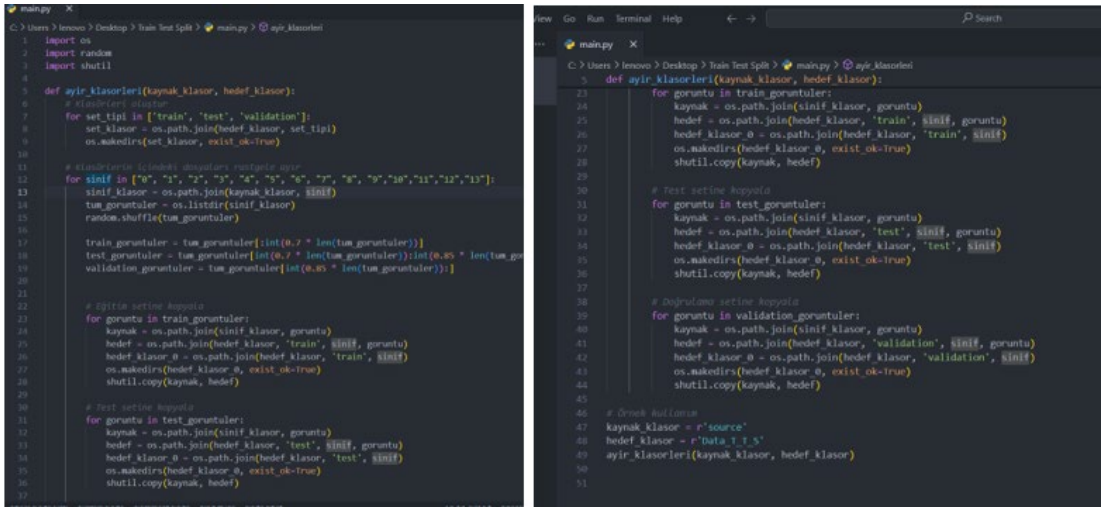
on Road Signs and Signals" adı altında yayınlanan bildiri ile uluslararası düzeyde trafik işaretlerinin standartlaştırılması konusunda önemli bir adım atılmıştır.
Bu süreç, trafik işaretlerinin anlamının ve kullanımının global olarak daha tutarlı hale gelmesine yardımcı olmuştur. Bu da sürücülerin uluslararası alanda daha güvenli ve bilinçli bir şekilde hareket etmelerini sağlamıştır.

3.2 Veri Seti

Trafik işaretlerinin makine öğrenmesi modelleriyle tanınmasını amaçlayan çalışmada kullanılmak üzere internet ortamında herkesin açık erişime sahip olduğu veri setleri bulunmaktadır. Bu tez çalışmasında kullanılan 3 farklı veri setlerinin içerisinden seçmiş olduğum sınıfları bir araya getirerek 14 adet sınıf oluşturdum. Kaggle üzerinden "Traffic Sign Images From Turkey (Erdem Yücesan,2020), Traffic Sign Dataset – Classification (ALURU V N M HEMATEJA, 2022), Traffic – Signs – Dataset (TUAN_AI, 2024) " veri setleri incelendi ve 14 adet sınıf belirlendi. Bu tez çalışması için oluşturulan veri setinde toplam 2716 adet giriş görüntü verisi ve 14 adet sınıftan oluşmaktadır. Veri setindeki tüm resimler png formatındadır. Tüm resimler 128x128 boyutunda yeniden boyutlandırılmıştır. Yapılan bu tez çalışmasında tüm modelleri eğitebilmek amacıyla ve en iyi sonuçlanan başarı oranlarını karşılamak amacıyla veri setinde bulunan resimlerin test, train ve velaidation data setleri olarak ayırım yapılmıştır. Bu ayrımın sebebi train datası genel veri setinin %60 %80 oranını kapsamaktadır. Modelin mümkün olduğunca çok örnek üzerinden öğrenmesini sağlayarak ağırlıkların (parametrelerin) belirlenmesini sağlar. Doğrulama (validation) genellikle veri setinin %10 %15 oranını kapsamaktadır. Modelin aşırı öğrenmesini engellemek hiperparametre ayarlarını optimize etmektir. Test verisi valid datası gibi %10 %15 oranını içermektedir. Model performansını değerlendirme ve genelleme yeteneğini değerlendirme için kullanılır. Eğitimde kullanılmayan veriler için modelin performansını değerlendirir. Test ve doğrulama setlerinin birbirine yakın olması, modelin tutarlılığını ve güvenilirliğini artırır. Bu yöntem, tezinizde model performansını değerlendirmek ve sonuçları objektif bir şekilde sunmak için kullanılmalıdır.
Şekil 3.1'de train veri seti içerisindeki sınıflara ait görseller yer almaktadır.

Şekil 3.1: Veri Seti Genel Görünümü

Train verisetinde 1890, test verisetinde 406 ve validaton veri setinde 420 adet veri bulunmaktadır. Bu veri setini Train, Test ve Validation olarak ayırmak için Python kütüphanelerinden yararlandım. Bu ayırma işlemini localimde Visual kod idesinde yapıldı.



```python
import os
import random
import shutil

def ayir_klasorleri(kaynak_klasor, hedef_klasor):
    # Klasörleri oluştur
    for set_tipi in ['train', 'test', 'validation']:
        set_klasor = os.path.join(hedef_klasor, set_tipi)
        os.makedirs(set_klasor, exist_ok=True)

    # Klasörlerin içindeki dosyaları rastgele ayır
    for sinif in ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9","10","11","12","13"]:
        sinif_klasor = os.path.join(kaynak_klasor, sinif)
        tum_goruntuler = os.listdir(sinif_klasor)
        random.shuffle(tum_goruntuler)

        train_goruntuler = tum_goruntuler[:int(0.7 * len(tum_goruntuler))]
        test_goruntuler = tum_goruntuler[int(0.7 * len(tum_goruntuler)):int(0.85 * len(tum_go
        validation_goruntuler = tum_goruntuler[int(0.85 * len(tum_goruntuler)):]


        # Eğitim setine kopyala
        for goruntu in train_goruntuler:
            kaynak = os.path.join(sinif_klasor, goruntu)
            hedef = os.path.join(hedef_klasor, 'train', sinif, goruntu)
            hedef_klasor_0 = os.path.join(hedef_klasor, 'train', sinif)
            os.makedirs(hedef_klasor_0, exist_ok=True)
            shutil.copy(kaynak, hedef)

        # Test setine kopyala
        for goruntu in test_goruntuler:
            kaynak = os.path.join(sinif_klasor, goruntu)
            hedef = os.path.join(hedef_klasor, 'test', sinif, goruntu)
            hedef_klasor_0 = os.path.join(hedef_klasor, 'test', sinif)
            os.makedirs(hedef_klasor_0, exist_ok=True)
            shutil.copy(kaynak, hedef)
```

```python
def ayir_klasorleri(kaynak_klasor, hedef_klasor):
        for goruntu in train_goruntuler:
            kaynak = os.path.join(sinif_klasor, goruntu)
            hedef = os.path.join(hedef_klasor, 'train', sinif, goruntu)
            hedef_klasor_0 = os.path.join(hedef_klasor, 'train', sinif)
            os.makedirs(hedef_klasor_0, exist_ok=True)
            shutil.copy(kaynak, hedef)

        # Test setine kopyala
        for goruntu in test_goruntuler:
            kaynak = os.path.join(sinif_klasor, goruntu)
            hedef = os.path.join(hedef_klasor, 'test', sinif, goruntu)
            hedef_klasor_0 = os.path.join(hedef_klasor, 'test', sinif)
            os.makedirs(hedef_klasor_0, exist_ok=True)
            shutil.copy(kaynak, hedef)

        # Doğrulama setine kopyala
        for goruntu in validation_goruntuler:
            kaynak = os.path.join(sinif_klasor, goruntu)
            hedef = os.path.join(hedef_klasor, 'validation', sinif, goruntu)
            hedef_klasor_0 = os.path.join(hedef_klasor, 'validation', sinif)
            os.makedirs(hedef_klasor_0, exist_ok=True)
            shutil.copy(kaynak, hedef)

# Örnek kullanım
kaynak_klasor = r'source'
hedef_klasor = r'Data_T_T_S'
ayir_klasorleri(kaynak_klasor, hedef_klasor)
```

Şekil 3.2: Datasetinin Train Test Validasyon Olarak Klasörlere Ayrılması

Python'ın os, random ve shutil kütüphanelerinden yararlanılmıştır.

-1 Os Kütüphanesi: , işletim sistemi ile etkileşimde bulunmak için kullanılan bir kütüphanedir. Bu kütüphane ile dosya ve dizin işlemleri yapılabilir.

-2 Random Kütüphanesi: Rastgele sayı üretimi ve rastgele seçim işlemleri için kullanılan bir kütüphanedir. Random Shuffle ise bir listenin elemanlarını rastgele sıralamak için kullanılır. Bu işlev, listedeki dosyaların rastgele seçilmesi ve bölünmesi için kullanılmıştır.

-3 Shutil Kütüphanesi: kütüphanesi, yüksek seviyeli dosya işlemleri için kullanılan bir kütüphanedir. Bu kütüphane, dosyaları kopyalama, taşıma ve silme gibi işlemler için kullanılır. Shutil.copy, bir dosyayı belirtilen kaynak yolundan hedef yoluna kopyalar. Bu işlev, dosyaların eğitim, test ve doğrulama dizinlerine kopyalanması için kullanılmıştır.

3.3 Eğitimde Kullanılan Yazılımlar ve Donanımlar

Makine öğrenmesi modellerini oluşturmak için Python dilini kullandım. Python'nın sağlamış olduğu birçok kütüphaneler sayesinde modellerimi geliştirdim. Bu tez çalışmasında geliştirme ortamı, Google'ın Colab Pro+ ortamında geliştirmeler yapılmıştır. Colan Pro+ sunmuş olduğu ortak CPU ve sanal depolama ortamından yararlandım. TensorFlow kütüphaneisnden Kerasi le geliştirme yapıldı. Keras ve TensorFlow Model oluşturma, eğitim ve değerlendirme işlemleri için kullanılır. Convolutional Neural Network (CNN) modeli oluşturulur ve eğitim verisi ile eğitilir. Keras, yüksek seviyeli bir yapay

sinir ağı API'sidir ve TensorFlow'un bir parçası olarak kullanılmaktadır. TensorFlow ise açık kaynaklı bir makine öğrenimi kütüphanesidir.

-1 Sequential: Keras'ta bir modeli katman katman oluşturmak için kullanılan bir model sınıfıdır.

-2 Conv2D: 2D konvolüsyon (evrişim) katmanıdır. Görüntü verisi üzerinde filtreler kullanarak özellik çıkarımı yapar.

-3 MaxPooling2D: Maksimum havuzlama katmanıdır. Uzamsal boyutları küçültmek ve hesaplama maliyetini azaltmak için kullanılır.

-4 GlobalAveragePooling2D: Global ortalama havuzlama katmanıdır. Özellik haritalarının ortalamasını alarak vektör boyutunu düşürür.

-5 Dense: Tam bağlantılı (fully connected) katmandır. Nöronlar arası tam bağlantı sağlar.

-6 Precision ve Recall: Modelin değerlendirilmesi için kullanılan doğruluk ve geri çağırma metrikleridir.

-7 ImageDataGenerator: Veri artırma ve ön işleme için kullanılır. Resim veri setlerini yükler ve dönüştürür.

-8 ReduceLROnPlateau ve ModelCheckpoint: Eğitim sırasında öğrenme oranını dinamik olarak ayarlamak ve en iyi model ağırlıklarını kaydetmek için kullanılan geri çağırma fonksiyonlarıdır.

Eğitimde kullanmış olduğum kişisel bilgisayarım üzerinde AMD Ryzen 5-5600H işlemci bulunduran ve 16GB RAM’e sahip bir bilgisayardır. Colab Pro+ üzerinde kodlar bu teknolojü üzerinde çalıştırılmıştır.

3.4 Yöntemler

Bu tez çalışmasında kullanılan tüm modeler CNN olup CNN’in alt kategorilerinden VGGNet16 ve ResNET50 mimarileri kullanılmış olup en iyi sonuç veren parametreler ile başarıları karşılaştırılmıştır.

Eğitim sonrası test verilerinin doğruluk oranlarını doğru bir şekilde karşılaştırma yapabilmek için veri setinin tüm modeler için aynı ön işleme işlemlerinden geçirilmiş olup model için gerekli olan parametreler değerlerinde ortak olacak şekilde ayarlanılmıştır.

-1 Tüm veriler boyut dönüşümü sonrası normalizasyon işlemine tabi tutulmuştur.

-2 Tüm modeler için epochs (eğitim döngü sayısı) 100 olarak belirlenmiş olup 10 eğitim döngüsünden sonra doğrulama değerinde iyileştirme göstermediği taktirde eğitimler durdurulmuştur. Deneme yanılma sonra epochs değerlerini sabit tutup batch sizelar (tek seferde modele verilecek örnek sayısı) ile değerlerini değiştirerek modellerin 100 epochs'da farklı batchsizelar ile eğitlip en iyi model performansı yakalanmıştır.

-3 Tüm modeler için aynı eğitim, doğrulama ve test verileri kullanılmıştır.

-4 Tüm modeler için optimizasyon yöntemi olarak Adam optimizasyonu metodu kullanılmıştır.

-5 Modellerin sınıflarına ait etiketler ve tahmin arasındaki çapraz entropi kaybını ölçmek amacıyla kayıp fonksiyon hesaplamasında Kategorical Cross Entropy kullanılmıştır.

Kullanılan CNN modellerde ortak parametreler:

-1 CNN : 100 epochs, Adam Optimizatör, Kategorical Cross Entropy

-2 RESNET50: 100 epochs, Adam Optimizatör, Kategorical Cross Entropy

-3 VGGNET16: 100 epochs, Adam Optimizatör, Kategorical Cross Entropy

3.5 CNN İle Eğitim Ve Sonuçları

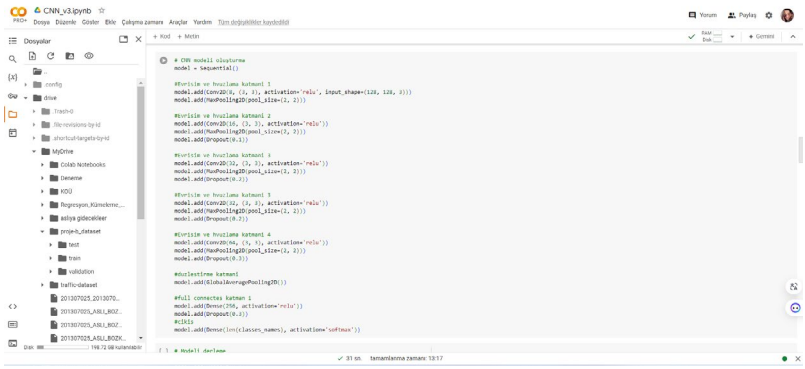
Modelin Katmanları

Modelin katmanlarının ve boyutlarının detayları:

1. Convolutional Katman 1:
    - Girdi: (128, 128, 3) (Renkli görüntüler)
    - Çıktı: (126, 126, 16)
    - Aktivasyon: ReLU
2. MaxPooling Katman 1:
    - Çıktı: (63, 63, 16)
3. Convolutional Katman 2:
    - Çıktı: (61, 61, 32)
    - Aktivasyon: ReLU
4. MaxPooling Katman 2:
    - Çıktı: (30, 30, 32)
5. Convolutional Katman 3:
    - Çıktı: (28, 28, 32)
    - Aktivasyon: ReLU
6. MaxPooling Katman 3:
    - Çıktı: (14, 14, 32)
7. Global Average Pooling Katmanı:
    - Çıktı: (32)
8. Dense Katman (Tam Bağlantılı Katman) 1:
    - Çıktı: (256)
    - Aktivasyon: ReLU
9. Çıkış Katmanı:
    - Çıktı: (14) (14 sınıf olduğu için)
    - Aktivasyon: Softmax

Model, Convolutional ve MaxPooling katmanlarından oluşan bir yapıya sahiptir ve ardından Global Average Pooling ile tamamen bağlı (Dense) katmanlara geçiş yapmaktadır. Global Average Pooling katmanı, CNN'in uzaysal boyutlarını (örneğin 14x14x32) tek bir vektör haline getirir (örneğin 32). Bu, modelin boyutunu küçültür ve tek bir vektörle ifade edilmesine olanak tanır.

Bu yapı, CNN modelinin çok katmanlı bir yapıya sahip olduğunu ve bu çok katmanlı yapının sonunda Global Average Pooling katmanı kullanarak uzaysal boyutların azaltıldığını ve ardından Dense katmanlar ile devam edildiğini gösterir.



Şekil 3.3: CCN Model Kod

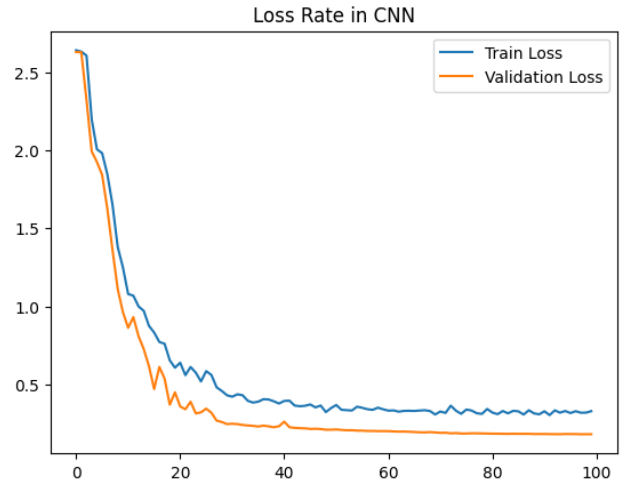
Oluşturulan model 406 adet test verisi üzerinde test edilmiştir. Aşağıda Şekil 3.4'de modelin test veri seti üzerindeki doğruluk oranı 0.96 olarak başarılı sonuç çıkmıştır.

```
[ ] # Model performansini degerlendirme
    evaluation = model.evaluate((test_generator), steps=len(test_generator))
    print("Test veri seti üzerinde modelin performansı: Loss={}, Accuracy={}, Precision={}, Recall={}".format(evaluation[0], evaluation[1], evaluation[2], evaluation[3]))

7/7 [==============================] - 3s 458ms/step - loss: 0.1722 - accuracy: 0.9606 - recall_2: 0.9606 - precision_2: 0.9606
Test veri seti üzerinde modelin performansı: Loss=0.17222703993320465, Accuracy=0.9605911374092102, Precision=0.9605911374092102, Recall=0.9605911374092102
```

Şekil 3.4 Model Değerlendirme Performansı Sonuç

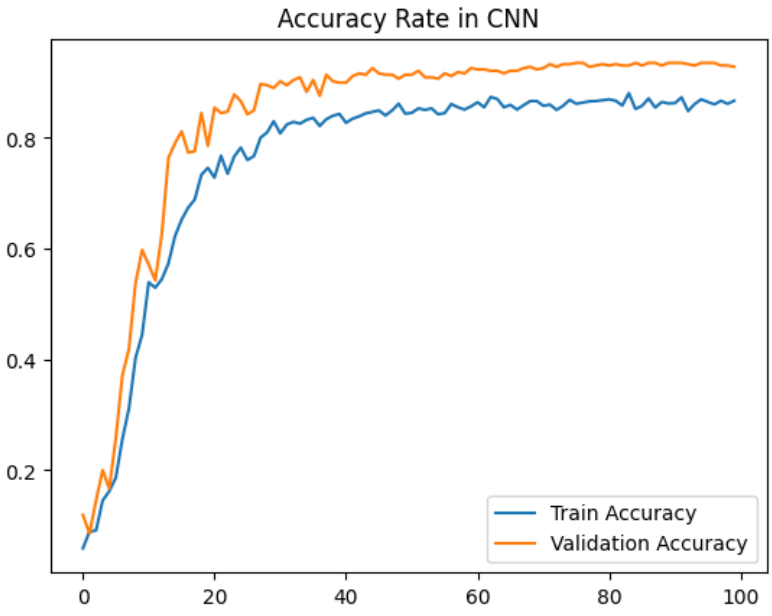
```
[ ] #gorsellestirme
    import matplotlib.pyplot as plt

    print(history.history.keys())
    plt.title("Loss Rate in CNN")
    plt.plot(history.history["loss"], label = "Train Loss")
    plt.plot(history.history["val_loss"], label = "Validation Loss")
    #loss ve accuracy Train uzerinde, val_loss ve val_accuracy Test uzerindeki sonuclar
    plt.legend()
    plt.show()
```

```
dict_keys(['loss', 'accuracy', 'recall_2', 'precision_2', 'val_loss', 'val_accuracy', 'val_recall_2', 'val_precision_2', 'lr'])
```



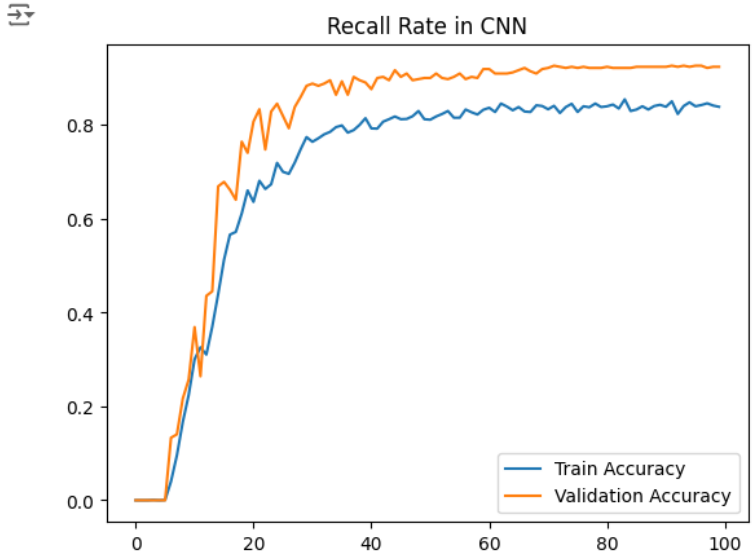
Şekil 3.5: CNN Modelde Train veValidaton Loss Fonksiyonu

```
[ ] plt.figure()
    plt.title("Accuracy Rate in CNN")
    plt.plot(history.history["accuracy"], label = "Train Accuracy")
    plt.plot(history.history["val_accuracy"], label = "Validation Accuracy")
    plt.legend()
    plt.show()
```



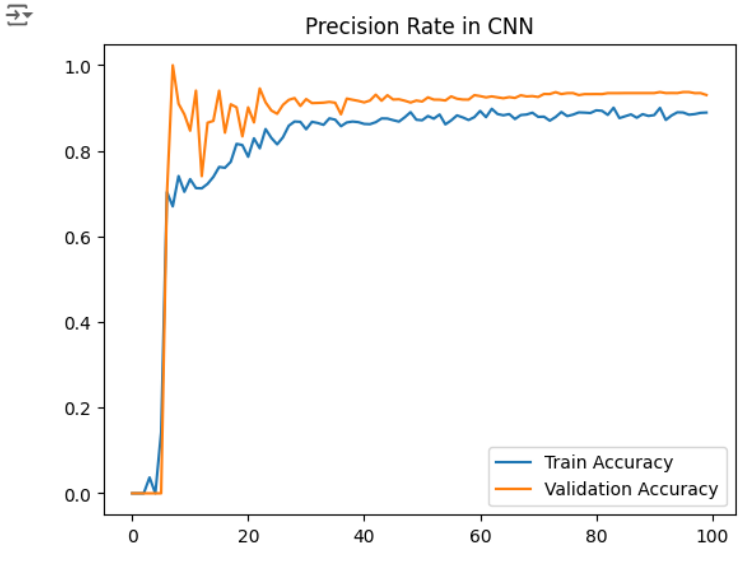
Şekil 3.5: CNN Modelde Train veValidaton Accuary Rate

```
[ ] plt.figure()
    plt.title("Recall Rate in CNN")
    plt.plot(history.history["recall_2"], label = "Train Accuracy")
    plt.plot(history.history["val_recall_2"], label = "Validation Accuracy")
    plt.legend()
    plt.show()
```



Şekil 3.6: CNN Modelde Train veValidaton Recall Rate

```
plt.figure()
plt.title("Precision Rate in CNN")
plt.plot(history.history["precision_2"], label = "Train Accuracy")
plt.plot(history.history["val_precision_2"], label = "Validation Accuracy")
plt.legend()
plt.show()
```



Şekil 3.7: CNN Modelde Train veValidaton Precision Rate

3.6 ResNET50 İle Eğitim Ve Sonuçları

ResNet50 modeli, derin bir evrişimli sinir ağı (CNN) modelidir ve kendisi zaten oldukça katmanlıdır. Eklenen katmanlar, ResNet50'nin çıktılarını alır ve bu çıktılar üzerine ek katmanlar ekleyerek modelin sınıflandırma işlevselliğini artırır.

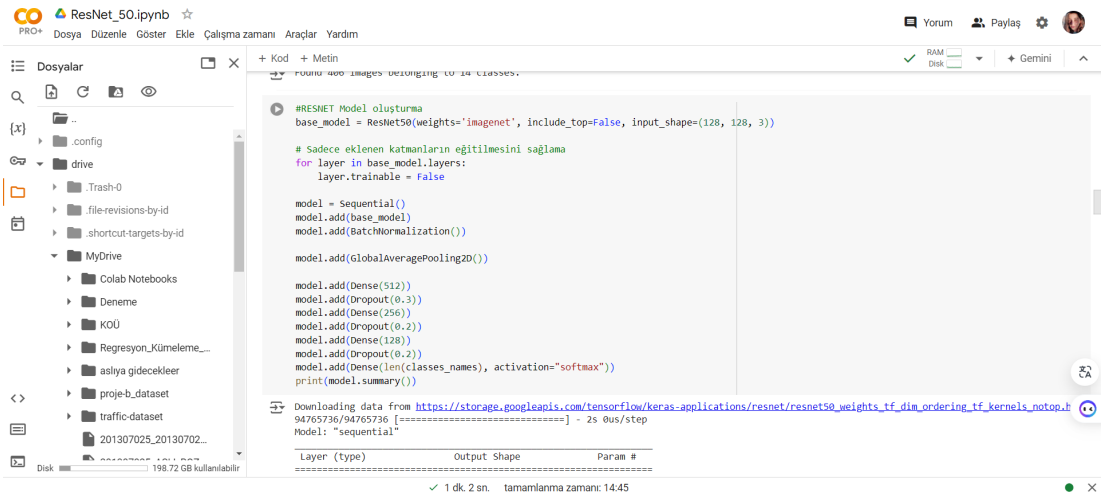
Dolayısıyla, verilen kodda modelin katman sayısı, ResNet50'nin katman sayısına ek olarak eklenen yeni katmanların sayısı kadardır. Bu durumda, ResNet50 modeli 50 katmanlı olduğu için, genişletilmiş modelin toplam katman sayısı 50'ye eklenen ek katmanlar kadar olacaktır. ResNet50 modelinin çıktısından sonra 5 ek katman eklenmiştir:

1. ResNet50 Baz Modeli (freeze edilmiş)
2. BatchNormalization Katmanı
3. GlobalAveragePooling2D
4. Dense (512 birim)
5. Dropout (0.3)
6. Dense (256 birim)
7. Dropout (0.2)
8. Dense (128 birim)
9. Dropout (0.2)
10. Dense (14 birim, Softmax aktivasyon)

Bunların yanı sıra, bir çıkış katmanı daha eklenmiştir. Dolayısıyla, eklenen 5 katmanın yanı sıra ResNet50 modelinin katmanlarıyla birlikte modelin toplam katman sayısı 55'tir. Toplam Katman Sayısı ResNet50'nin katmanları + 5 ek katman

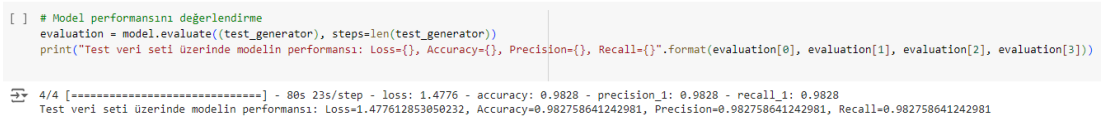
Girdi Boyutu (128, 128, 3) Bu boyut, modelin giriş görüntülerinin (yükseklik, genişlik, kanal sayısı) şeklini belirtir.

Çıkış Katmanı Softmax aktivasyon fonksiyonuna sahip, 14 birimden oluşan Dense katmanı bu katman, modelin 14 sınıftan birini tahmin etmesini sağlar.



Şekil 3.8: Resnet50 Model

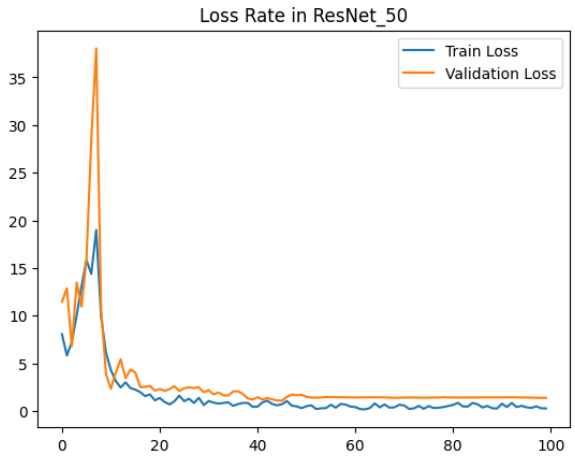
Oluşturulan model 406 adet test verisi üzerinde test edilmiştir. Aşağıda Şekil 3.9'de modelin test veri seti üzerindeki doğruluk oranı 0.98 olarak başarılı sonuç çıkmıştır.



Şekil 3.9 Model Değerlendirme Performansı Sonuç

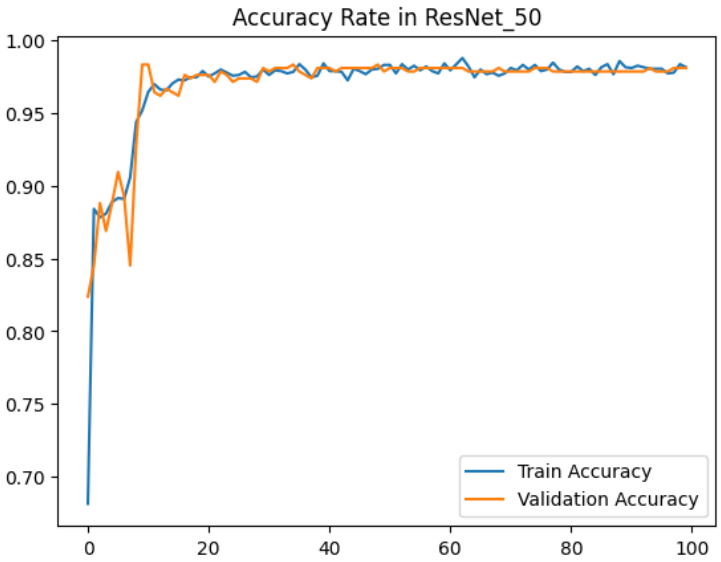
```
#GÖRSELLEŞTİRME
import matplotlib.pyplot as plt
print(history.history.keys())
plt.title("Loss Rate in ResNet_50")
plt.plot(history.history["loss"], label = "Train Loss")
plt.plot(history.history["val_loss"], label = "Validation Loss")
plt.legend()
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'precision_1', 'recall_1', 'val_loss', 'val_accuracy', 'val_precision_1', 'val_recall_1', 'lr'])
```



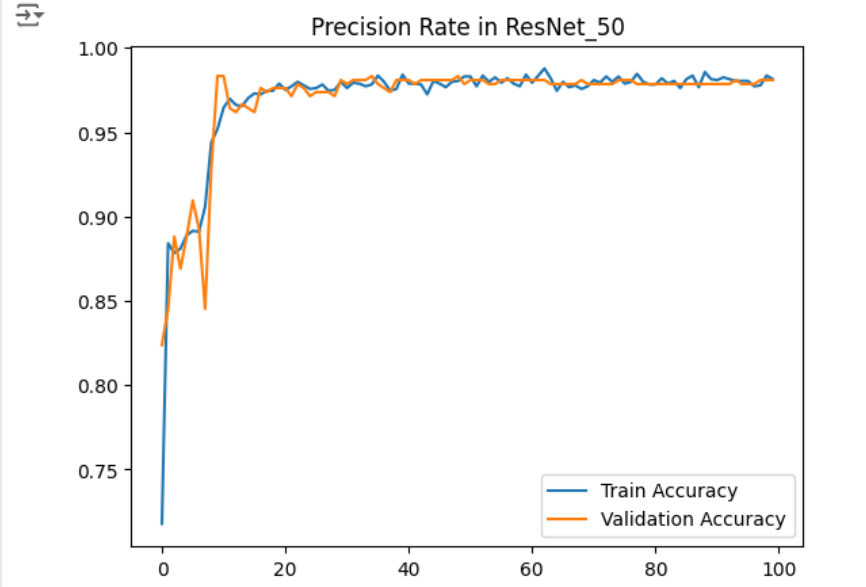
Şekil 3.10: Resnet50 Modelde Train veValidaton Loss Rate

```
plt.figure()
plt.title("Accuracy Rate in ResNet_50")
plt.plot(history.history["accuracy"], label = "Train Accuracy")
plt.plot(history.history["val_accuracy"], label = "Validation Accuracy")
plt.legend()
plt.show()
```



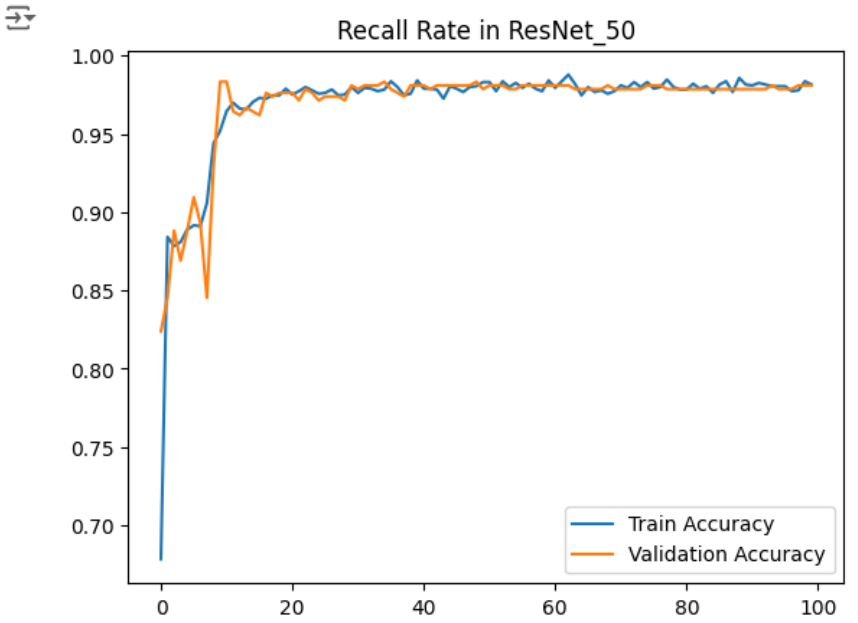
Şekil 3.11: Resnet50 Modelde Train veValidaton Accuary Rate

```
plt.figure()
plt.title("Precision Rate in ResNet_50")
plt.plot(history.history["precision_1"], label = "Train Accuracy")
plt.plot(history.history["val_precision_1"], label = "Validation Accuracy")
plt.legend()
plt.show()
```



Şekil 3.12: Resnet50 Modelde Train veValidaton Precision Rate

```
[ ] plt.figure()
    plt.title("Recall Rate in ResNet_50")
    plt.plot(history.history["recall_1"], label = "Train Accuracy")
    plt.plot(history.history["val_recall_1"], label = "Validation Accuracy")
    plt.legend()
    plt.show()
```



Şekil 3.13: Resnet50 Modelde Train veValidaton Recall Rate

3.7 VGGNET16 İle Eğitim Ve Sonuçları

Model toplamda aşağıdaki katmanlardan oluşur:

VGG16 Base Model: 13 Convolutional + 5 Pooling katmanı (18 katman)

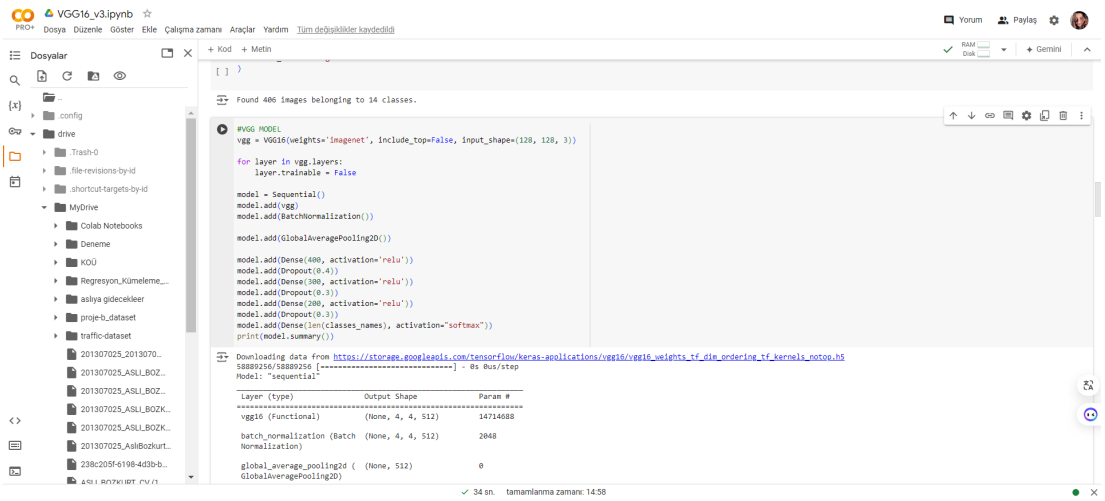
Eklenen Katmanlar:
1 BatchNormalization
1 GlobalAveragePooling2D
3 Dense
3 Dropout
Toplamda, VGG16 modeli hariç 8 katman eklenmiştir. Ancak VGG16 modeli de dahil edildiğinde toplam katman sayısı artar.

Modelin Katmanlı Yapısı
Bu model, katmanlı yapıdan tek katmanlı yapıya dönüştürülmemiştir. Aksine, bu modelde VGG16 gibi çok katmanlı bir özellik çıkarıcı kullanılmış ve bunun üzerine çeşitli tam bağlantılı (dense) katmanlar eklenmiştir. Yani model, daha derin ve daha karmaşık hale getirilmiştir, tek katmanlı bir yapıya dönüştürülmemiştir.

Modelin Yapısı
Model: "sequential"

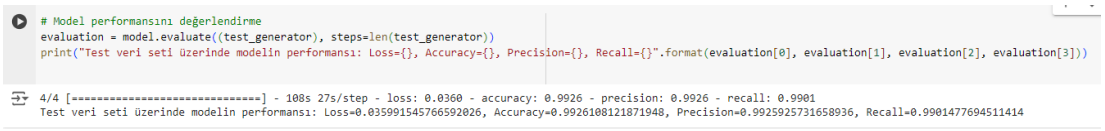
```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Functional)           (None, 4, 4, 512)         14714688
_________________________________________________________________
batch_normalization (BatchNo (None, 4, 4, 512)         2048
```

```
_________________________________________________________________
___
global_average_pooling2d (Gl (None, 512)              0
_________________________________________________________________
___
dense (Dense)                (None, 400)              205200
_________________________________________________________________
___
dropout (Dropout)            (None, 400)              0
_________________________________________________________________
___
dense_1 (Dense)              (None, 300)              120300
_________________________________________________________________
___
dropout_1 (Dropout)          (None, 300)              0
_________________________________________________________________
___
dense_2 (Dense)              (None, 200)              60200
_________________________________________________________________
___
dropout_2 (Dropout)          (None, 200)              0
_________________________________________________________________
___
dense_3 (Dense)              (None, 14)               2814
=================================================================
==========
Total params: 15,105,250
Trainable params: 388,662
Non-trainable params: 14,716,588
```



Şekil 3.14: VGG16 Model

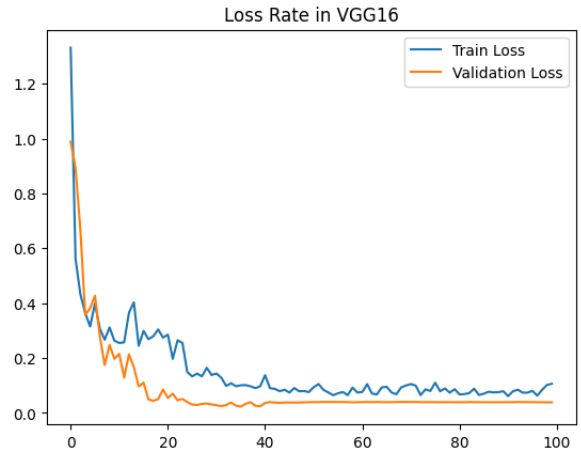
Oluşturulan model 406 adet test verisi üzerinde test edilmiştir. Aşağıda Şekil 3.15'de modelin test veri seti üzerindeki doğruluk oranı 0.99 olarak başarılı sonuç çıkmıştır.



Şekil 3.15 Model Değerlendirme Performansı Sonuç

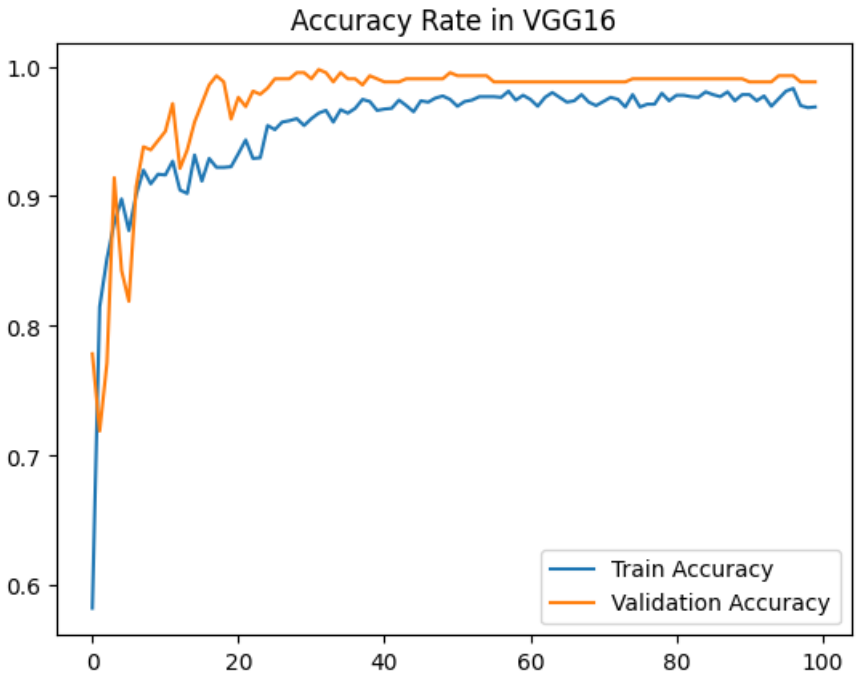
```
#result visualize
import matplotlib.pyplot as plt
print(history.history.keys())
plt.title("Loss Rate in VGG16")
plt.plot(history.history["loss"], label = "Train Loss")
plt.plot(history.history["val_loss"], label = "Validation Loss")
plt.legend()
plt.show()
```

```
dict_keys(['loss', 'accuracy', 'precision', 'recall', 'val_loss', 'val_accuracy', 'val_precision', 'val_recall', 'lr'])
```



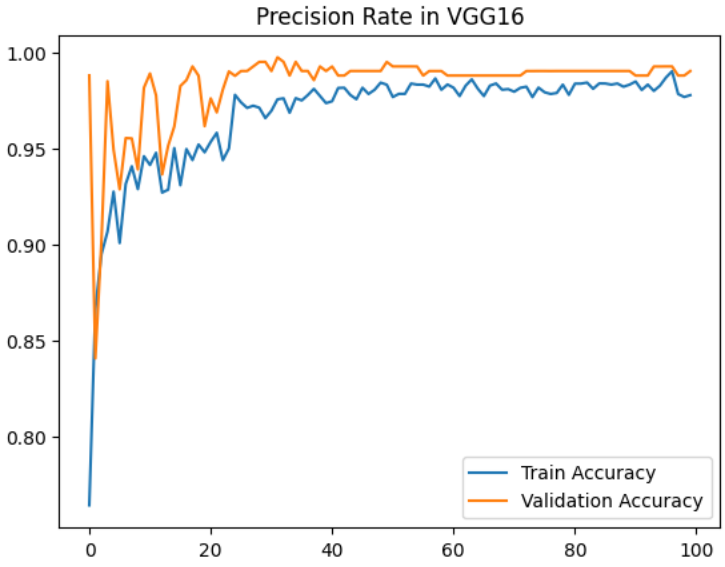
Şekil 3.16: VGG16 Modelde Train veValidaton Loss Rate

```
plt.figure()
plt.title("Accuracy Rate in VGG16")
plt.plot(history.history["accuracy"], label = "Train Accuracy")
plt.plot(history.history["val_accuracy"], label = "Validation Accuracy")
plt.legend()
plt.show()
```



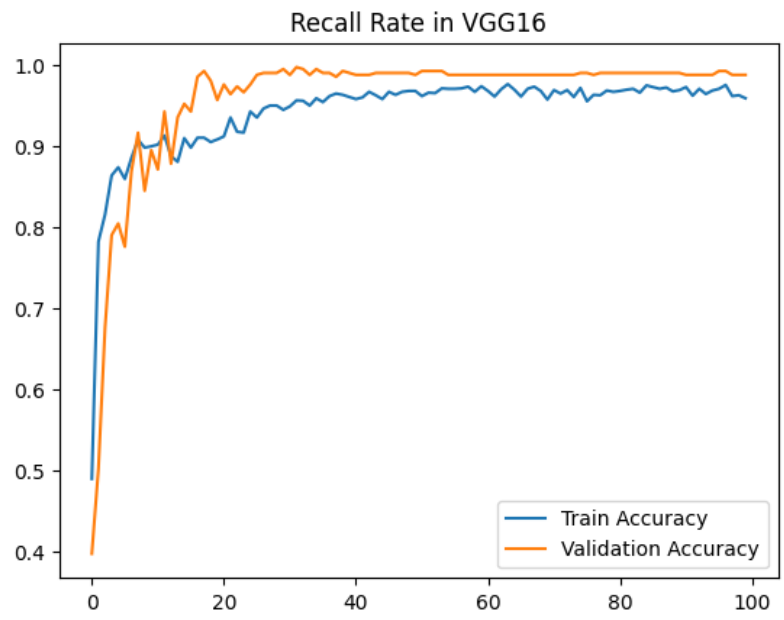
Şekil 3.17: VGG16 Modelde Train veValidaton Accuary Rate

```
plt.figure()
plt.title("Precision Rate in VGG16")
plt.plot(history.history["precision"], label = "Train Accuracy")
plt.plot(history.history["val_precision"], label = "Validation Accuracy")
plt.legend()
plt.show()
```



Şekil 3.18: VGG16 Modelde Train veValidaton Precision Rate

```
[ ] plt.figure()
    plt.title("Recall Rate in VGG16")
    plt.plot(history.history["recall"], label = "Train Accuracy")
    plt.plot(history.history["val_recall"], label = "Validation Accuracy")
    plt.legend()
    plt.show()
```

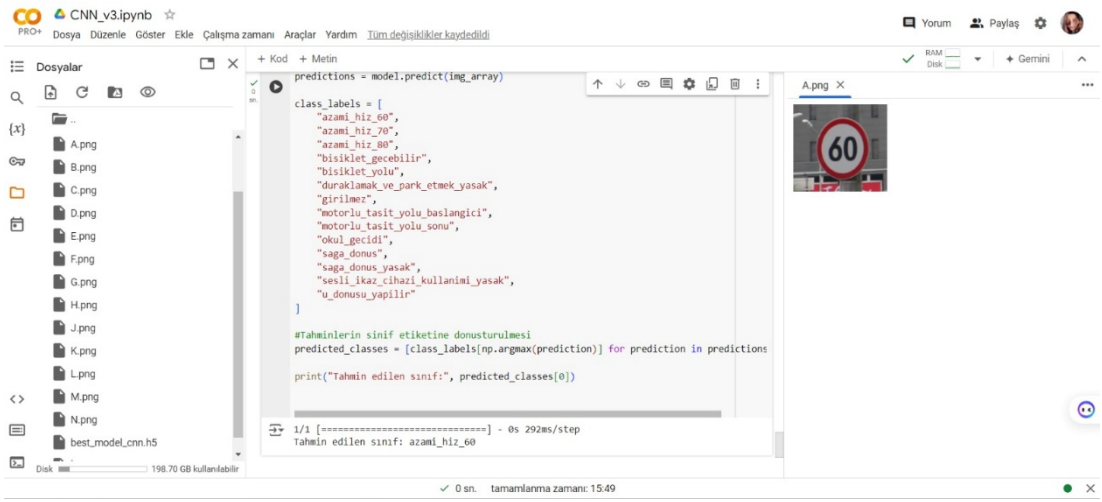


Şekil 3.19: VGG16 Modelde Train veValidaton Recall Rate

4. SONUÇLAR VE ÖNERİLER

Bu tez çalışmasında CNN yapay sinir ağı modeli kullanılmış olup CNN modellerden Resnet50 ve VGGNet16 kullanılmıştır. 408 adet test verisi üzerinden en iyi performans sağlayacak şekilde parametre ayarlamaları yapılıp en iyi başarılı sonuçlar edlde edilmiştir.

Şekil 4.1'de görüldüğü üzere test verilerinden herhangi bir sınıftan çekilen resmin doğru tahmin edildiği görülmektedir.



Şekil 4.1: Test Verisetini Tahmin Etme Başarısını Test Etme

Yapılan Deneyler Sonucunda en iyi tahmin başarısını VGG16 %99, ResNet50 %98 ve CNN % 96 olarak sonuçlar sıralanmaktadır.

Oluşturulan modellerin tahmin başarısını iyileştirmek için, model parametreleri en uygun sonuca ulaşana kadar tekrar ayarlanabilir veya derin ağ mimarilerinin katman sayıları artırılabilir. Ancak, bu tür işlemler eğitim zamanı maliyetinin artmasına neden olabilir. Aşağıda modellerin eğitim süreleri verilmiştir.

CNN: 128 dakika

ResNET50: 132 dakika

VGG16: 720 dakika

Bu tez çalışmasında ilerleyen zamanlarda daha fazla veri ve sınıflarla çalışılarak sürücü destek sistemi geliştirilebilir. Bu sistem araç içerisine yerleştirilen kamera ile bu kameradan alınan görüntüleri kullanarak görüntü taraması yapıp yol bilgileri sürücüye yansıtılması yapılabilir.

KAYNAKLAR

[1] Derin Öğrenme Algoritmaları ile Trafik İşaret ve Levhalarının Tanımlanması, Ahmet yavuz 2021

[2] https://hilalgozutok.medium.com/evri%C5%9Fimli-sinir-a%C4%9Flar%C4%B1-convolutional-neural-networks-cnn-e61470e9bdb1

[3] https://dergipark.org.tr/en/download/article-file/596690 Yapay Sinir Ağları ve Yapay Zekâ'ya Genel Bir Bakış Kadir Öztürk1,*, Mustafa Ergin Şahin1

[4] https://suhedacilek.medium.com/resnet-residual-network-nedir-49105e642566

[5] https://tr.d2l.ai/chapter_convolutional-modern/resnet.html

[6] https://evrimagaci.org/makineler-dusunebilir-mi-alan-turingin-1950-tarihli-hesaplama-makineleri-ve-zeka-makalesinin-turkce-tam-cevirisi-17573

[7]https://medium.com/@umitkabann22/makine-%C3%B6%C4%9Frenmesi-76df665fc78d

[8] https://medium.com/academy-team/deep-learningde-pre-trained-transfer-learning-modeller-nas%C4%B1l-%C3%A7al%C4%B1%C5%9F%C4%B1r-41969977b95e

[9] https://ayyucekizrak.medium.com/derin-%C3%B6%C4%9Frenme-i%C3%A7in-aktivasyonfonksiyonlar%C4%B1n%C4%B1nkar%C5%9F%C4%B1la%C5%9Ft%C4%B1r%C4%B1lmas%C4%B1-cee17fd1d9cd

[10]https://medium.com/databulls/yapay-sinir-a%C4%9Flar%C4%B1nda-aktivasyon-fonksiyonlar%C4%B1-11002b8ac522

[11]https://medium.com/kodcular/yapay-sinir-a%C4%9Flar%C4%B1nda-aktivasyon-fonksiyonlar%C4%B1-%C3%A7e%C5%9Fitleri-14c2ab782866

[12] https://polen.itu.edu.tr/bitstream/11527/681/1/8040.pdf

[13]https://medium.com/@k.ulgen90/makine-%C3%B6%C4%9Frenimi-b%C3%B6l%C3%BCm-3-4b160df1f4c8

[14] https://www.veribilimiokulu.com/yapay-sinir-agiartificial-neural-network-nedir/

[15]https://mkotan.sakarya.edu.tr/sites/mkotan.sakarya.edu.tr/file/EKO469_VM_H12_Siniflandirma3.pdf

[16]https://acikerisim.sakarya.edu.tr/xmlui/bitstream/handle/20.500.12619/76861/T01687.pdf?sequence=1&isAllowed=y

[17]https://slideplayer.com/slide/14855329/

[18]https://frightera.medium.com/alexnet-vggnet-inception-ve-resnet-nedir-bddc7482918b

[19]https://medium.com/yapay-zeka-makine-%C3%B6%C4%9Frenmesi-derin-%C3%B6%C4%9Frenme/derin-%C3%B6%C4%9Frenme-mimarileri-ce3c16a22ffc

[20]https://openai.com/chatgpt/

ÖZGEÇMİŞ

2002 yılında İstanbul'da doğdu. İlk ve orta öğrenimini İstanbul'da tamamladı. 2016 yılında girdiği Güngören Anadolu Lisesi'nden 2020 yılında mezun olmuştur. 2020 yılında girdiği Kocaeli Üniversitesi Teknoloji Fakültesi Bilişim Sistemleri Mühendisliği'nde öğrenci.