

**YALOVA ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESİ**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**- BİTİRME TEZİ –**

**FASHİON MNİST VERİSETİNİN EVRİŞİMSEL SİNİR AĞLARI İLE SINIFLANDIRILMASI**

**Adı SOYADI**

Zeyna BAAJ

**Bitirme Tezi Danışmanı: prof.dr. Müfit ÇETİN**

**YALOVA, 2020**

**YALOVA ÜNİVERSİTESİ**

**MÜHENDİSLİK FAKÜLTESI**

**BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ**

**FASHİON MNİST VERİSETİNİN EVRİŞİMSEL SİNİR AĞLARI İLE SINIFLANDIRILMASI**

**Öğrenci No Adı SOYADI**

140101070 Zeyna BAAJ

1. Bitirme Tezi Danışmanı : **Doç. Dr. MÜFIT ÇETIN**

2. Jüri Üyesi : **Dr. Öğrt.Ü. ADEM TUNCER**

3. Jüri Üyesi : **Dr.Öğrt.Ü. OSMAN HİLMİ KOÇAL**

Bitirme Tezinin Dönemi: 2019 – 2020 Güz Yarıyılı

**Table of Contents**

[KISALTMALAR LİSTESİ iii](#_Toc29457807)

[ŞEKİL LİSTESİ iv](#_Toc29457808)

[Özet v](#_Toc29457809)

[Abstract vi](#_Toc29457810)

[1. GİRİŞ 7](#_Toc29457811)

[1.1. Sinir Ağlarının Tarihçesi 7](#_Toc29457812)

[2. YAPAY SİNİR AĞLARI 12](#_Toc29457813)

[2.1. Aktivasyon Fonksiyonları 12](#_Toc29457814)

[2.2. Çok Katmanlı Yapay Sinir Ağları Yapısı 13](#_Toc29457815)

[2.3. Hata Hesaplama 14](#_Toc29457816)

[2.4. Optimizasyon Metodları 15](#_Toc29457817)

[3. KONVOLÜSYONEL SİNİR AĞLARI 17](#_Toc29457818)

[3.1. Girdi Katmanı 17](#_Toc29457819)

[3.2. Konvolüsyon Katmanı 17](#_Toc29457820)

[3.3. Ortaklama Katmanı 20](#_Toc29457821)

[3.4. Tam Bağlantılı Katman 21](#_Toc29457822)

[3.5. Konvolüsyon (Convolution) Aritmetiği 22](#_Toc29457823)

[3.5.1. Sıfır dış boşluk (padding) ve bir adım değeri 22](#_Toc29457824)

[3.5.2. Sıfır dış boşluk (padding) ve birden farklı adım değeri 22](#_Toc29457825)

[3.5.3. Sıfırdan farklı dış boşluk ve aralık sayısı 23](#_Toc29457826)

[3.5.4. Girdi boyutunun korunması 24](#_Toc29457827)

[3.6. Ortaklama (Pooling) Aritmetiği 24](#_Toc29457828)

[3.7. Fashion-MNIST veri seti 25](#_Toc29457829)

[4. UYGULAMA 26](#_Toc29457830)

[4.1. Veri Seti 26](#_Toc29457833)

[4.2. CNN Modeli 26](#_Toc29457834)

[4.3. Tahmin 28](#_Toc29457835)

[DEĞERLENDİRME VE ÖNERİLER 30](#_Toc29457836)

[KAYNAKLAR 32](#_Toc29457837)

# **KISALTMALAR LİSTESİ**

|  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| |  |  | | --- | --- | | **Kısaltma** | **Açıklama** | | BT | Benzetimli Tavlama | | DE | Diferansiyel Evrim | | GA | Genetik Algoritma | | HA | Harmonik Arama | | KSA | Konvolüsyonel Sinir Ağları | | µO | Mikrokanonikal Optimizasyon | | MT | Mikrokanonikal Tavlama | | PSO | Parçacık Sürü Optimizasyonu | | SGD | Stokastik Gradyan Azaltma | | TPE | Tree Parzen Estimator | | YA | Yerel Arama (Local Search) | | YSA | Yapay Sinir Ağı | |  | |
|  |  |

# **ŞEKİL LİSTESİ**

**Şekil 1.1** Perceptron yapısı 10

**Şekil 1.2** XOR problemi 10

**Şekil 1.3** SVM 11

**Şekil 1.4** El yazısı tanıma veri seti için kullanılan konvolüsyonel sinir ağı mimarisi 12

**Şekil 2.1** Aktivasyon fonksiyonları 14

**Şekil 2.2** Çok katmanlı YSA yapısı 15

**Şekil 2.4** İki gizli katmanlı bir YSA parametreleri 18

**Şekil 3.1** Renkli girdiler için kanal, genişlik ve yükseklik kavramlarının gösterimi 20

**Şekil 3.2** Girdi için 3 kanaldan oluşan filtrenin uygulanması (aralık = 1, dış boşluk = 0) 21

**Şekil 3.3** Birden fazla filtre için konvolüsyon işlemin uygulanması ve çıktılar (aralık = 1, dış boşluk = 0) 22

**Şekil 3.4** 3x3 Filtre boyutu ve 1 aralık (stride) değeri için maksimum ve ortalama ortaklama çıktıları 23

**Şekil 3.5** 2x2 Filtre boyutu ve 2 adım (stride) değeri için maksimum ve ortalama ortaklama çıktıları 23

**Şekil 3.6** 4x4 boyutunda bir girdi 3x3’lük bir filtre ile konvolüsyon işlemine sokuluyor (dış boşluk = 0, stride = 1, çıktı boyutu = 2x2) 24

**Şekil 3.7** 5x5 boyutunda bir girdi 3x3’lük bir filtre ile konvolüsyon işlemine sokuluyor (dış boşluk = 0, stride = 2, çıktı boyutu = 2x2) 25

**Şekil 3.8** 5x5 boyutunda bir girdiye 1x1’lik kenar ekleniyor ve 3x3’lük bir filtre ile konvolüsyon işlemine sokuluyor. (dış boşluk = 1, stride = 2, çıktı boyutu = 3x3) 25

**Şekil 3.16** Fashion-MNIST veri seti sınıflar ve örnek görüntüler 27

# **Özet**

Bilgisayarlı görü çalışmaları, günümüzde en çok ilgi duyulan ve üzerinde çalışma yapılan yapay zeka alanlarından biridir. Bilgisayarların insanlar gibi görüntüleri algılamasını, sınıflandırabilmesini ve yorumlayabilmesini sağlamak amacıyla geliştirilen özel derin öğrenme mimarileri bulunmaktadır. Bunlardan en çok kullanılan ve bu çalışmada da bahsedilecek olan mimari konvolüsyonel sinir ağları mimarisidir. Konvolüsyonel sinir ağları, bilgisayarlı görü çalışmalarında popüler olarak kullanılan ve başarılı sonuçlar elde edilebilen özelleşmiş bir derin öğrenme yöntemidir. Derin öğrenme yöntemleri karşılaşılan problemlerin zorluğu nedeniyle yüksek hesaplama maliyetlerine neden olabilmektedir. Hesaplama maliyetinin düşürülmesi güçlü donanımların kullanılmasına, oluşturulan konvolüsyonel sinir ağı topolojilerindeki toplam parametre sayısının azaltılmasına ve konvolüsyonel sinir ağlarındaki hiper- parametreler için seçilen değerlere bağlıdır. Bu yüzden konvolüsyonel sinir ağlarında hiper-parametre optimizasyonu çalışmaları, ağın başarısını arttırmaya çalışırken, hesaplama maliyetini de düşük tutmaya çalışmaktadırlar. Bu tez çalışmasında ilk olarak daha önce konvolüsyonel sinir ağlarının optimize edilmesi için gerçekleştirilen optimizasyon çalışmaları incelendi. İncelenen çalışmalarda, konvolüsyonel sinir ağlarında hiper-parametrelerin optimizasyonu için sıklıkla üst-sezgisel algoritmaların ve istatistik tabanlı model bazlı algoritmaların kullanıldığı gözlemlendi. Özellikle Genetik Algoritma, Parçacık Sürü Optimizasyonu, Diferansiyel Evrim, Rastgele Arama ve Bayes Optimizasyonu gibi yöntemlerin, incelenen çalışmalarda sıklıkla kullanıldığı gözlemlendi. Bu çalışmalar başarı açısından incelendiklerinde genetik algoritma ve parçacık sürü optimizasyonu yöntemlerinin genel olarak hiper-parametre optimizasyonu gerçekleştirmeyen çalışmalara göre başarılı ve rekabetçi sonuçlar verdiği görüldü. Yapılan tez çalışmasında kullanılacak veri setleri, seçilecek optimizasyon yöntemi, hiper-parametreler ve değer aralıklarının belirlenmesi için incelenen çalışmalarda kullanılan veri setleri, hiper-parametreler ve bu hiper- parametreler için seçilen değer aralıkları göz önünde bulunduruldu. Yapılan çalışmalarda, farklı çalışmaları karşılaştırmak için elimizde parametre sayısı ve hesaplama zamanı bilgileri bulunmadığından sadece doğruluk oranı bilgisi performans karşılaştırması için kullanıldı. Daha önce yapılan bu çalışmalardan farklı olarak bu tez çalışmasında “Mikrokanonikal Optimizasyon” olarak adlandırılan bir yöntem kullanıldı. Seçilen optimizasyon yöntemi kullanılarak farklı boyutlarda konvolüsyonel sinir ağları oluşturuldu ve oluşturulan konvolüsyonel sinir ağlarının hiper- parametreleri optimize edilmeye çalışıldı. Seçilen optimizasyon algoritmasının çalışması sırasında üretilen konvolüsyonel sinir ağları, bilgisayarlı görü çalışmalarında sıklıkla kullanılan MNIST, FashionMNIST, EMNIST (Balanced, Digits, Letters) ve CIFAR10 veri setleri üzerinde test edildi. Elde edilen sonuçlar, hiç hiper-parametre optimizasyonu gerçekleştirmeyen ve state-of-the-art olarak adlandırılan çalışmalar ile doğruluk oranı ve parametre sayısı gibi değerler üzerinden karşılaştırıldı. Ek olarak, önerilen sezgisel yöntemin performansı, Bayesçi model tabanlı bir optimizasyon yöntemi olan Tree Parzen Estimator yöntemiyle karşılaştırılmıştır. Elde edilen sonuçlara bakıldığında, Konvolüsyonel sinir ağları için belirlenmesi gereken birçok hiper-parametre olmasına rağmen seyreltme oranı, filtre sayısı, öğrenme oranı ve yığın boyutu gibi hiper-parametrelerin oluşturulan modellerin başarısında önemli bir katkısı olduğu çıkarımına ulaşıldı.

# **Abstract**

Computer vision is probably the most widely studied sub-area of artifical intelligence which has been drawing considerable interest of many researchers for years. There are special deep learning architectures developed to enable computers to perceive, classify and interpret images as humans. Convolutional neural networks are the most popular deep learning methods that can be used successfully in computer vision studies. Deep learning methods may result in high computational costs due to the difficulty of the problems encountered. This computational cost can only be reduce by careful selection of hyperparameters of the convolutional neural networks and the computational time can also be reduce by the use of powerful equipment. Therefore, in the studies that try to optimize hyperparameters in convolutional neural networks, the researchers try to increase the success rate of the network while at the same time try to keep the computational cost as low as possible. In this thesis, firstly a detailed literature review on the studies that perform hyperparameter optimization has been given. It has been observed that heuristic algorithms and statistics model based algorithms are among the most widely used methods for hyper-parameter optimization in convolutional neural networks. In particular, Genetic Algorithms, Particle Swarm Optimization, Differential Evolution, Random Search and Bayes Optimization methods are the most frequently used approaches. When we compare these methods in terms of their success rates, we see that the studies in which genetic algorithms and particle swarm optimization methods are used were able to achieve greater results than the studies that did not perform hyper-parameter optimization in general. In order to determine the optimization method to be used in the study along with the hyper-parameters and their value ranges, we benefited the studies in the literature. Moreover, the datasets used in this study are selected among the most widely used datasets in the literature. Most of the studies in the literature do not provide sufficient information about the number of parameters of the network and the computational time, therefore we took in the account accuracy as the performance measure. In this study, Microcanonical Optimization which is previously known in different areas was but not used in this concepts has been applied fort he hyperparameter optimization of convolutional neural networks. By this method, different network architectures has been created and the hyper-parameters of the network is optimized. The convolutional neural networks generated during the optimization process are trained on the MNIST, FashionMNIST, EMNIST (Balanced, Digits, Letters) and CIFAR10 datasets, which are the most frequently used datasets in computer vision studies. The accuracy results are compared to the state-of-the-art architectures in which no hyper-parameter optimization has been performed. In addition, the performance of proposed heuristic method has been compared to Tree Parzen Estimator method which is a Bayesian model based optimization method. The results suggest that among the many hyperparameters dropout rate, feature map count, learning rate and batch size are among the most important parameters that directly affect the success of the networks.

# **1. GİRİŞ**

İnsan gibi düşünen, insana benzer, sorunlara insanların bakış açısıyla çözüm üretebilen genel bir zeka oluşturmak bilgisayar bilimleri için her zaman önemli bir çalışma konusu olmuştur. Bunların temelinde daha zeki sistemler tasarlamak, sadece belirli bir iş için uzmanlaşmış sistemler yerine insan gibi öğrenebilen, çözüm üretebilen, bulunduğu ortama ve şartlara uyum sağlayabilen sistemleri oluşturma isteği yatmaktadır. Bu sistemleri geliştirebilmek için en başta insan beyninin modellenmesi gerekmektedir. İnsan beynini modelleyebilmek için ise insan beyninin biyolojik olarak nasıl çalıştığını anlamak gerekmektedir.

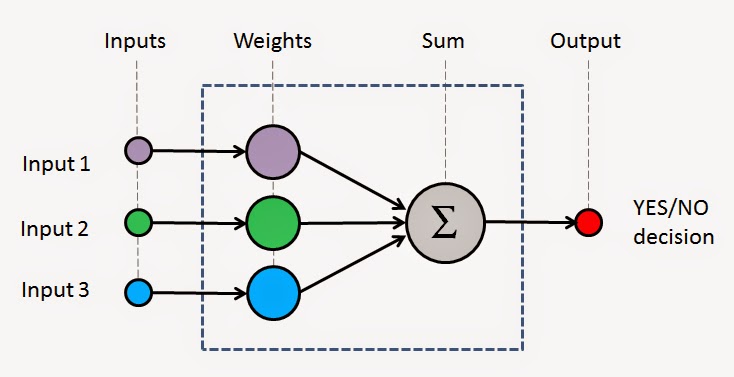
## Sinir Ağlarının Tarihçesi

1940’lı yıllardan beri daha zeki sistemler geliştirebilmek adına yapılan çalışmalar mevcuttur. İlk olarak 1943 yılında S. McCulloch ve Walter H. Pitts tarafından yayınlanan “A Logical Calculus of The Ideas Immanent In Nervous Activity” isimli makale ile bu çalışmalar başlamıştır. Bu çalışmada insan beyninde öğrenmeyi sağlayan sinir ağlarından esinlenilmiştir. Beyin fonksiyonları, sinir aktiviteleri için mantıksal bir analiz yapılmış ve matematiksel modeller ile açıklanmıştır. Sinirsel aktiviteler ve bunlar arasındaki ilişkiler (ağırlıklar), önermeli mantık yolu ile iyileştirilebilir denilmiş ve günümüzde kullanılana benzer bir yapay sinir ağı modeli gerçekleştirilmiştir. Bu modelde nöronlar ve nöronlar arasındaki ilişkileri ifade etmek için ağırlıklar kullanılmıştır. Fakat günümüzdeki yaklaşımdan farklı olarak ağırlıkların öğrenilemeyeceği fikri savunulmuştur.

1958 yılında F. RosenBlatt tarafından “The perceptron: a probabilistic model for information storage and organızatıon ın the brain” çalışması yayınlanmıştır. Bu çalışma temel olarak 3 temel soruya yanıt aramaktadır :

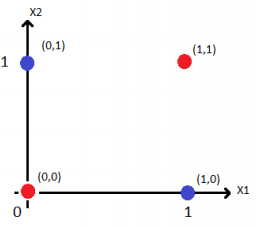
* Biyolojik sistem tarafından fiziksel dünyadaki olaylar nasıl algılanır veya tespit edilir?
* Elde edilen bilgiler hangi formda saklanır veya hatırlanır?
* Depolamada veya bellekte yer alan bilgiler tanıma ve davranışları nasıl etkiler?

Bu yaklaşım da insanda bulunan sinir ağlarının çalışma yapısından esinlenilerek geliştirilmiştir. Aktivasyon fonksiyonu ve eşik değeri (threshold) kavramları bu çalışmayı önemli kılan noktalardandır. Belirli bir eşik değerine göre nöron ve çıktıların güncellenmesini ve karar vermeyi sağlayan bir yapı oluşturulmuştur. Bu yapı ile ilk defa öğrenme kavramı da konuşulmaya başlanmıştır. Basit mantıksal operatörler için bu yapının başarılı sonuçlar vermesi insanların bu modelin hatasız olduğu fikrine kapılmalarına neden olmuştur. Ta ki XOR problemi ile karşılaşılıncaya kadar. Yapılan bu çalışma günümüz yapay sinir ağlarının oluşumu için önemli bir temel oluşturmuştur.



**Şekil 1.1** Perceptron yapısı

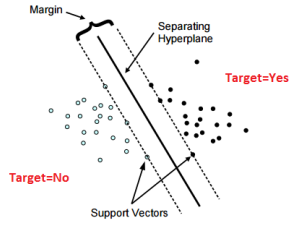
1960’lı yılların sonuna doğru gelindiğinde, 1969 yılında M. Minsky ve S. Papert tarafından yayınlanan “Perceptrons” isimli çalışma ile F.Rosenblatt tarafından ortaya koyulan perceptron yapısının (şekil 1.1) doğrusal olmayan problemler için başarısız olduğu ortaya konulmuştur. Perceptron yapısının doğrusal ayrılabilen problemler için başarılı olduğu bilinmekteydi. Fakat Perceptron yapısı XOR problemi (şekil 1.2) gibi doğrusal olmayan bir problem için uygulandığında, bu yapının başarısız olduğu görülmüştür. Yani perceptron yapısı ile doğrusal olmayan problemlerin sonuçlarının iki boyutlu uzayda tek bir doğrusal çizgi ile ayrılamayacağı anlaşılmıştır. Bu durum yapay zeka çalışmalarındaki 9 yıllık (1960 - 1969) altın çağı sonlandırmıştır. Aynı zamanda bu problem 1986 yılına kadar çözülememiş ve yapay zeka için karanlık çağın (1969 – 1986) başlangıcına neden olmuştur.



**Şekil 1.2** XOR problemi

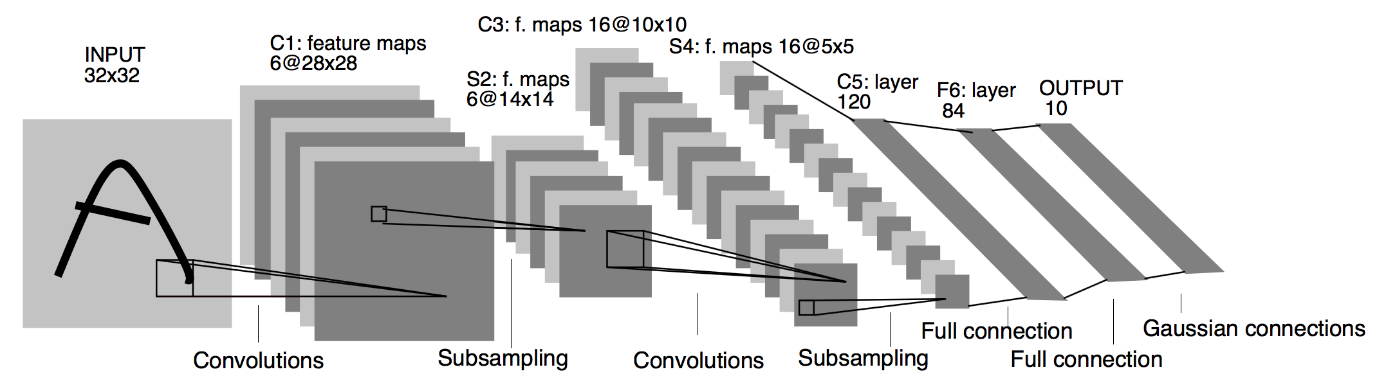
1986 yılına gelindiğinde bugün ki derin öğrenme modellerinin de temellerini oluşturacak bir yaklaşım ortaya atıldı. 1986 yılında G. Hinton, D. Rumelhart ve R. Williams tarafından yayınlanan “Learning representations by back-propagations errors” isimli çalışma yapay sinir ağları için yeni bir öğrenme metodu getirdi. Yeni metod geri yayılım olarak isimlendirildi. Daha önce kullanılan perceptron yapısında kullanılan katman sayısı arttırılarak Çok Katmanlı Perceptron (Multi Layer Perceptron) yapısı elde edildi. Bu yapının eğitilmesi, ağırlıkların doğru bir şekilde güncellenmesi için geri yayılım algoritması kullanıldı. Geri yayılım algoritması, ağda bulunan nöronlar arasındaki bağlantıları, ağırlıkları gerçek çıktı vektörü ile hesaplanan çıktı vektörü arasındaki farkın minimize edilmesini sağlayacak şekilde günceller. Girdi ve çıktı katmanları arasındaki katmanlar gizli katman (hidden layer) olarak isimlendirilir. Öğrenme prosedürü, verilen girdi için istenen çıktıyı elde edebilmek adına gizli birimlerin hangi durumlarda aktif olacağına karar vermelidir. Hatanın minimuma indirgenmesi için katmanlar arası ağırlıkların doğru bir şekilde güncellenmesi gerekir. Geri yayılım algoritması ağırlıkların güncellenmesini, seçilen optimizasyon yöntemine bağlı olarak her bir örnekten sonra ya da belirli örnek topluluklarından (batch) sonra hesaplanan hataya göre yineleyerek yapar. Ağırlıkların güncellenmesi hataların türevi alınacak şekilde yapıldığından ve katman sayısı ya da katmanlardaki nöron sayılarının fazla olmasından dolayı yüksek işlem gücü gerektirir. Fakat geri yayılım algoritması ve çok katmanlı perceptron yapısı doğrusal olmayan, lineer olarak ayrılamayan problemlerin çözümü için başarılı sonuçlar elde etmiştir ve yapay zeka çalışmalarını tekrar ivmelendirmiştir.

Çok katmanlı perceptron yapısının ortaya çıktığı dönemde yeterli işlem gücünün olmaması yapay sinir ağlarının gelişimini olumsuz yönde etkilemiştir. Bu dönemde daha az işlem gücü gerektiren ve analitik yöntemlere dayanan yeni bir öğrenme yaklaşımı ortaya çıkmıştır. 1995 yılında V. Vapnik ve C. Cortes tarafından “Support Vector Networks” isimli çalışma yayınlanmıştır. Bu çalışma, iki gruplu sınıflandırma problemleri için yeni bir öğrenme makinesi olarak tanıtılmıştır. Sınıflandırma işlemi vektörel işlemler sonucu elde edilen karar destek çizgileri ile yapılmaktadır. Şekil 1.3 ‘te karar destek vektör makinesi gösterilmiştir.



**Şekil 1.3** SVM

1990’lı yıllarda Yann LeCun tarafından AT & T Bell Labs’ta yürütülen el yazısı tanıma projesi , bugün kullandığımız anlamda konvolüsyonel sinir ağlarının temelini oluşturmuştur. Konvolüsyonel Sinir Ağları (KSA), katmanlarının en az bir tanesinde matris çarpımı yerine konvolüsyon işleminin kullanıldığı çok katmanlı yapay sinir ağlarının bir türüdür. Aynı zamanda derin öğrenme ağlarının bilgisayarlı görü çalışmaları için özelleşmiş bir mimarisidir. KSA’lar 3. bölümde daha detaylı anlatılacaktır. Yann LeCun, ağın eğitilmesi için kendisinin kurduğu Le-Net 5 ağını kullanmıştır. Bu çalışma başarılı sonuçlar elde etse de yeterli hesaplama gücünün elde edilememesi ve ağların eğitimi için yeterli miktarda eğitim verisinin olmaması bu çalışmaların bir süre daha köşede beklemesine neden olmuştur. Şekil 1.4 ‘te Yann LeCun tarafından yapılan çalışmada kullanılan Le-Net mimarisi gösterilmiştir.



**Şekil 1.4** El yazısı tanıma veri seti için kullanılan konvolüsyonel sinir ağı mimarisi

2000’li yıllarda Geoffrey Hinton tekrar sahneye çıktı ve sinir ağlarını, çok fazla gizli katman içeren derin öğrenme ağlarına dönüştürdü. Derin öğrenme ağları etkili sonuçlar vermesine rağmen gerektirdiği yüksek işlem gücü nedeniyle hala etkin olarak kullanılamamaktaydı. 2006 yılında Geoffrey Hinton, ağırlıkların, parametrelerin ve katman sayısının çok fazla olduğu derin öğrenme ağlarının açgözlü katmanlı ön eğitim metodu kullanılarak başarılı bir şekilde eğitildiğini kanıtlamıştır.

Gelişen teknoloji ile birlikte derin öğrenme ağlarının eğitilmesi için gerekli olan işlem gücüne nispeten ulaşılmaya başlandı. Bulut hizmetlerinin artması, paralel işlem gücü, ekran kartlarının çekirdek sayıları ve hızları, mobil cihazların ve sosyal medya kullanımının artması ile elde edilen büyük veri setleri derin öğrenme çalışmalarını hızlandırdı. 2010 yılında Stanford Üniversitesi’nde Fei-Fei Li’nin başında bulunduğu grup “ImageNet” adı verilen milyonlarca etiketlenmiş resim içeren veri setini oluşturdu ve paylaştı.

Büyük ölçekli resimlerin tanınması için geliştirilen mimari ve algoritmaların yarıştığı LSVRC yarışması düzenlenmeye başlandı. İlki 2010 yılında düzenlenen yarışmanın ilk yıllarında %28 ve %26 doğruluk oranları (accuracy) elde edildi. 2012 yılında ise Alex Krizhevsky, Ilya Sutskever ve Geoffrey E. Hinton tarafından geliştirilen mimari Top-1 için 37.5% ve Top-5 için 17% doğruluk değerleri elde etti. Bu ifadelerde Top-1, tahmin olasılığı en yüksek olan sonucun gerçek sonuç ile doğru olarak eşleşme olasılığını, Top-5 ise tahmin olasılığı en yüksek olan 5 sonuç içerisinde gerçek sonucun geçme olasılığını temsil etmektedir (Yani tahmin ettiğim 5 sonuç içerisinde gerçek sonuç ile eşleşen bir sonuç var mı?). Elde ettiği bu başarı ve getirmiş olduğu yenilikçi yaklaşımlar derin öğrenme ağları ile yapılan çalışmaların tekrar hız kazanmasını sağlamıştır.

Bu çalışmanın ayrıntıları “ImageNet Classification with Deep Convolutional Neural Networks” adıyla yayınlanmıştır. Bu çalışmanın en önemli noktalarından biri de çok daha fazla çekirdek sayısına sahip olan ve verilerin paralel bir şekilde hesaplanabilmesine imkan sağlayan GPU’lardır. GPU’lar kullanılarak derin öğrenme modelleri çok daha hızlı bir şekilde eğitilebilmektedir. Bu durum da daha büyük modellerin oluşturulmasına, eğitilmesine ve daha başarılı sonuçların elde edilmesine olanak sağlamaktadır.

Günümüzde derin öğrenme çalışmaları bilgisayar bilimlerinin en sıcak konularından biri haline gelmiştir. Özellikle bilgisayarlı görü tarafında yapılan çalışmalar ve geliştirilen mimariler sonucunda elde edilen başarı oranları 2015 yılında insan hata oranının altına inmiştir. Yani eğitilen modeller insanlardan daha iyi bir doğrulukla görüntüleri sınıflayabilmektedir.

Bu tez çalışmasında yapay sinir ağlarının yapısı, sinir ağlarının öğrenme süreci, konvolüsyonel sinir ağlarının yapısı, konvolüsyonel sinir ağlarında kullanılan hiperparametreler ve bu hiper-parametrelerin optimizasyonu için kullanılan yöntemler anlatılmıştır. Aynı zamanda optimizasyon çalışması yapılan konvolüsyonel sinir ağları ile optimizasyon çalışması yapılmayan konvolüsyonel sinir ağları karşılaştırılmıştır.

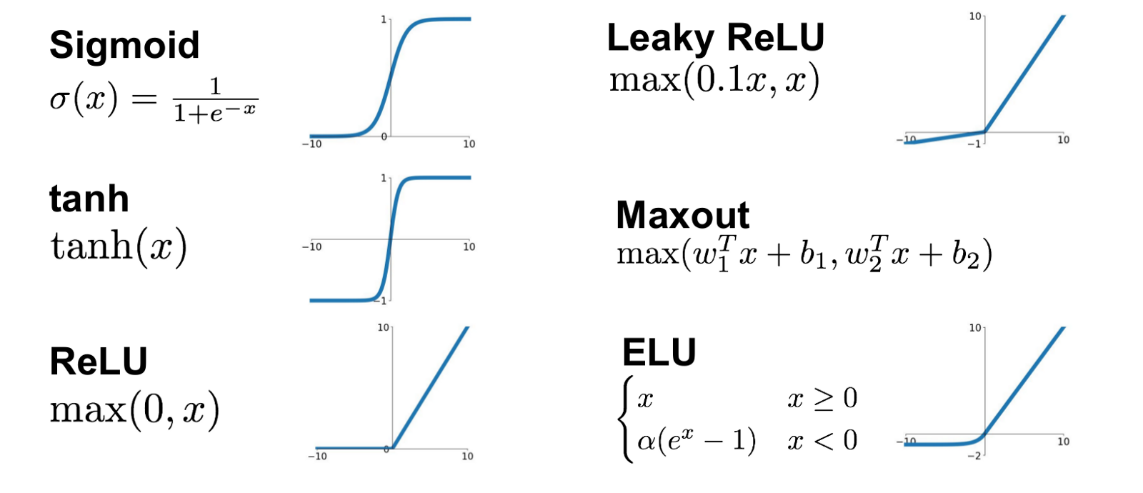
Karşılaştırma için kullanılan yöntemler anlatıldı ve daha önce bu alanda yapılan çalışmalar ile performans karşılaştırması gerçekleştirildi. Sonuç olarak konvolüsyonel sinir ağlarını oluştururken hiper-parametrelerin seçiminde dikkat edilmesi gereken noktalara, günümüzde sıklıkla kullanılan yöntemlere ve ileride kullanılabilecek metodolojilere değinildi.

# **2. YAPAY SİNİR AĞLARI**

İnsan beyninin nasıl çalıştığı yıllarca bu alanda çalışan sinirbilimciler için önemli bir araştırma konusu olmuştur. Yapılan bu araştırmalar boyunca insan beynini oluşturan sinir ağlarının birçok modeli oluşturulup, insan beyninin nasıl öğrendiği sorusuna cevap aranmaya çalışılmıştır. İnsan beyninin nasıl anladığını öğrenmek için sinir ağlarının çalışma yapısının da öğrenilmesi gerekmektedir. Yapılan bu çalışmalardaki amaç yalnızca insan beyninin veya sinir ağlarının nasıl çalıştığını anlamak değil aynı zamanda insan beyni gibi öğrenebilen yapay sistemler, sinir ağları tasarlayabilmektir. İnsan beyninin öğrenme özelliğini makinelere kazandırmak için geçmişten günümüze birçok çalışma yapılmıştır. Bu alanda bilgisayar bilimciler sinir ağlarının oluşturulan modellerini göze alarak yapay sinir ağlarını oluşturmuşlardır. Sinir ağının temel bileşenleri olan nöron, sinaps, dentrit ve aksonlar yapay sinir ağlarında karşılık gelecek şekilde nöron, girdi, çıktı ve ağırlıklar olarak adlandırılmış ve ilk yapay sinir ağı modelleri oluşturulmuştur. İlk başlarda sadece girdi, girdilere ait ağırlıklar, toplama fonksiyonu, aktivasyon fonksiyonu ve tek bir çıktıdan oluşan bu daha basit sistem algılayıcı (perceptron) olarak adlandırılmıştır.

## Aktivasyon Fonksiyonları

Aktivasyon fonksiyonları, gelen girdilere karşılık oluşacak olan çıktıları belirler. Seçilen aktivasyon fonksiyonu, yapay sinir ağını oluşturan elemanların doğrusal yapısının, doğrulsal olmayan yapıya dönüştürülmesini sağlar. Yapılan dönüşüm sonucunda oluşan doğrusal olmayan yapı, sinir ağının türevlenmesi kolay hale getirilmesini sağlar. Türev alma işlemi ile girdiler için tanımlanan ağırlıkların hata değerine etkisi hesaplanır, sonuca göre ağırlıklar hata değerini azaltacak şekilde güncellenir. Kolay türevlenebilir fonksiyonlar, sinir ağlarının daha hızlı hesaplamalar yapması için gereken avantajı sağlar. Bu nedenle, sinir ağlarında hesaplama zamanının azaltılması için türevlenmesi kolay olan aktivasyon fonksiyonları seçilmelidir. Sıklıkla kullanılan aktivasyon fonksiyonları şunlardır: Sigmoid, tanjant hiperbolik, doğrultulmuş lineer ünite ve sızdırılmış doğrultulmuş lineer ünite aktivasyon fonksiyonları (Şekil 2.1).

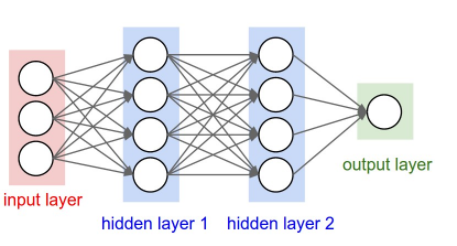


**Şekil 2.1** Aktivasyon fonksiyonları

## Çok Katmanlı Yapay Sinir Ağları Yapısı

Çok katmanlı yapay sinir ağları temel olarak 3 katmandan meydana gelir. Sıralı bir yapay sinir ağı modelini oluşturan bu katmanlar: Girdi Katmanı (Input Layer), Gizli Katmanlar (Hidden Layers) ve Çıktı Katmanıdır (Output Layer).

**Girdi Katmanı (Input Layer):** Yapay sinir ağında gelen bilgiler girdi katmanında temsil edilir. Veri setine göre değişen özelliklerin her biri farklı bir düğüm olarak girdi katmanında temsil edilir. Her bir girdinin ağırlık değeri vardır. Bu girdiler gizli katmandaki düğümler ile bu ağırlıklar aracılığı ile bağlıdır. Girdiler için belirlenen ağırlık değerleri yapay sinir ağında o özelliğin önemini, ağırlığını belirtmektedir.



**Şekil 2.2** Çok katmanlı YSA yapısı

**Gizli Katmanlar (Hidden Layers):** Gizli katmanlar, girdi katmanından gelen bilgilerin işlenip bir çıktıya dönüştürüldüğü katmanlardır. Çıktıya dönüştürme işlemi yapay sinir ağının ağırlık değerleri kullanılarak gerçekleştirilir. Gizli katman sayısı problemin zorluğuna göre çeşitlilik gösterebilmektedir. Her bir gizli katman belirlenen sayıda nörona (unit) sahiptir. Nöronlar öğrenilen özelliklerin tutulduğu birimler olarak düşünülebilir. Her bir nöron sonuca olan etkisine göre belirlenen ağırlıklara sahiptir. Bu ağırlıkların aldığı değerlere göre gizli katmandaki nöronların sonuca olan etkisi gözlenebilir. Şekil 2.2 ’de gösterilen yapıda 2 gizli katman bulunmaktadır ve her iki gizli katman da 4 adet nöron içermektedir.

**Çıktı Katmanı (Output Layer):** Verilen girdi değerine ait çıktı değeri veya değerlerinin tutulduğu katmandır. Bu katmanda, seçilen hata hesaplama fonksiyonu kullanılarak, sinir ağı tarafından üretilen çıktı değeri ile beklenen çıktı değeri arasındaki fark hesaplanır. Hata hesaplama fonksiyonu sonucunda elde edilen hata değeri ağın performansının değerlendirilmesi, istenen sonuçlara ne kadar uzak olduğumuzun belirlenmesi için kullanılır. Elde edilen hata değerine göre ağırlıkların güncellenmesi gerekmektedir. Ağırlıkların güncellenmesi, yani yapay sinir ağının öğrenme işlemi, seçilecek optimizasyon fonksiyonu ile gerçekleştirilir. Ağırlıkların güncellenmesi için farklı yöntemler kullanılabilir. Bazı yöntemler şunlardır;

* **İleri Beslemeli Ağlar:** Akışın girdi katmanından çıktı katmanına doğru gerçekleştiği ağ yapısıdır. Verilen x girdi vektöründen ŷ çıktısının elde edildiği yapay sinir ağlarında, x girdi vektörü sinir ağı için ilk bilgiyi sağlar. Girdinin sağladığı bu ilk bilgi gizli katmanlarda bulunan gizli düğümlere, birimlere yayılır ve ŷ çıktısı üretilir. Bu yönteme ileri yayılım, bu yöntemi kullanan ağlara ise ileri beslemeli ağlar denir. Bir katmandan gelen nöronların çıktıları, diğer katmandaki nöronlara ağırlıklar aracılığıyla girdi olarak verilir. Denklem 2.1'de, daha önce Şekil 1.1’de gösterilen perceptron ile n özelliğe sahip bir x girdisi için (i Є 1,..n) çıktı değerinin nasıl hesaplandığı gösterilmiştir; w ağırlıkları, b yanlılık (bias) değerini ve f seçilen aktivasyon fonksiyonunu temsil etmektedir. Yanlılık değeri aktivasyon fonksiyonunun sağa veya sola kaydırılmasını sağlar. Aynı zamanda yanlılık değeri yapay sinir ağlarında aşırı öğrenmeyi (overfitting) engellemek adına hesaplanan çıktıya eklenen sabit veya rastsal bir değerdir.

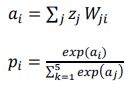
2.1)

Yapay sinir ağlarında öğrenme adımlarının hızlandırılması için ağırlık başlatıcı (weight initiator) yönteminin doğru seçilmesi çok önemlidir. Sinir ağında bulunan ağırlıkların başlatılması için kullanılan birçok yöntem bulunmaktadır. En çok kullanılan yöntemlerden biri ağdaki ağırlıkların rastgele değerler ile başlatılmasıdır. Ağırlıklara atanan rastgele değerler belirli aralıklarla ([-1,1]) sınırlandırılabilir. Bunun dışında Gauss dağılımı ve varyans değerlerini kullanarak ağırlıkların daha akıllıca bir şekilde başlatılmasını sağlayan Xavier (Glorot) gibi ağırlık başlatıcılar yapılan çalışmalarda sıklıkla kullanılmaktadır.

* **Geri Beslemeli Ağlar:** Akışın yalnızca ileriye doğru değil aynı zamanda geriye doğru da olabildiği ağ yapısıdır. Ağırlıkların güncellenmesi geriye yada ileriye doğru gerçekleştirilebilir. Geri besleme yöntemiyle, hesaplanan hatanın ağın geriye kalan kısımlarına dağıtılması sağlanır. Kısmi türev ve zincir kuralı kullanılarak ağırlıkların güncellenmesi gerçekleştirilir. Kullanılan bu yöntem ile hesaplanan toplam hataya, her bir ağırlığın etkisi hesaplanır ve bu hesaplamalara göre ağırlıklar güncellenir.

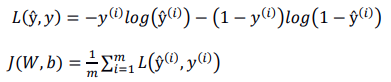
## Hata Hesaplama

YSA’larda üretilen çıktı değerlerinin, beklenen çıktı değerlerine ne kadar uzak olduğunun belirlenebilmesi için hata hesaplama işlemi gerçekleştirilir. Gerçekleştirilen bu işlemlerde YSA’nın maliyetinin (cost) ölçülmesi için sınıf sayısı ve problemin tipine bağlı olarak (sınıflandırma, regresyon); Çapraz entropi (Cross Entropy) ve ortalama kare hata (Mean Squared Error), kök ortalama kare hata (Root Mean Squared Error) gibi yöntemler sıklıkla kullanılır.

Softmax yöntemi, olasılık temelli bir fonksiyondur. Problem tipine bağlı olarak Sınıflandırma işleminin gerçekleştirileceği problemin içerdiği sınıf sayısı kadar çıktı birim oluşturulur. Oluşturulan değerler için üretilen olasılıklar ile en çok benzerliğe sahip olan düğüm belirlenmeye çalışılır. Örneğin 5 sınıftan oluşan bir problem için 5 düğüm oluşturulur ve çıktı katmanından önceki katmanda bulunan düğümler ile ağırlıklar denklem 2.2’ da görüldüğü üzere çarpılır ve böylece her bir i sınıfı için ai aktivasyon değerleri hesaplanır. Daha sonra her bir i sınıfı için 0 ile 1 arasındaki olasılık değeri, pi, denklem 2.3' de gösterildiği hesaplanır ve en yüksek pi değerine sahip sınıf tahmin edilen sınıf olarak belirlenir.

2.2)

2.3)

Olasılık değerlerinin hesaplanmasının ardından "Cross Entropy" yöntemi ile her bir girdi için hata hesaplanması denklem 2.4'e göre gerçekleştirilir (ŷ: hesaplanan değer, y: gerçek değer). Toplam hata (maliyet) denklem 2.5’a göre hesaplanır (m: örnek sayısı).

2.4)

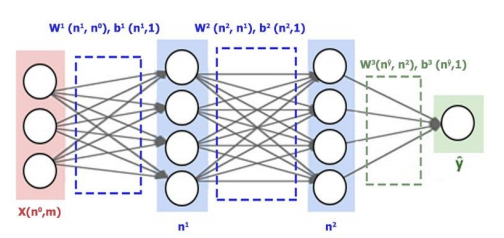
2.5)

Yapay sinir ağlarının sınıflandırma başarılarını arttırmak için bazen düzenlileştirme (regularization) yöntemleri kullanılabilir. Maliyet fonksiyonunun sonuna bir ceza eklenerek yapay sinir ağı modelleri genelleştirilmeye çalışılır, yani eğitilen yapay sinir ağının sadece eğitim örnekleri için değil daha önce hiç görmediği örnekler için de başarılı sonuçlar vermesi hedeflenir. Aşırı öğrenmeyi (overfitting) engellemek amacıyla en sık kullanılan düzenlileştirme yöntemleri şunlardır: L2 Düzenlileştirme (Ridge Regression), L1 Düzenlileştirme (Lasso Regression) ve Seyreltme (Dropout).

## Optimizasyon Metodları

Yapay sinir ağlarında, ağırlıkların doğru bir şekilde güncellenmesi öğrenme işlemi için çok önemlidir. Hata hesaplandıktan sonra elde edilen hataya göre ağırlıklar güncellenir. Ağırlıkların güncellenmesi için Geri Yayılım Algoritması (BackPropagation) kullanılır. Geri Yayılım Algoritması ile sinir ağındaki her bir ağırlığın, hesaplanan hataya olan etkisini hesaplamak için gradyan tabanlı Stokastik Gradyan Azaltma , RMS-Prop, Adam ve Adadelta metodları sıklıkla kullanılmaktadır.

**Gradyan Azaltma (Gradient Descent):** Yapay sinir ağlarında hata optimizasyonu için kullanılan yöntemlerin en temelinde Gradyan azaltma algoritması bulunmaktadır. Bu optimizasyon yöntemlerinin en temelindeki amaç, toplam hatayı geriye doğru yayarak, her bir ağırlığın toplam hataya olan etkisinin hesaplanmasıdır. Bu şekilde ağın optimizasyonu, öğrenme işlemi gerçekleştirilir. Seçilen optimizasyon yöntemine göre optimizasyon süreçleri değişse de kullanılan optimizasyon yöntemlerinin temelinde türev, zincir alma kuralı ve geri yayılım bulunmaktadır.



**Şekil 2.4** İki gizli katmanlı bir YSA parametreleri

Sinir ağının toplam maliyeti/hatası (J) denklem 2.4’a göre hesaplanır. Gradyan azaltma metodunda hesaplanan toplam hata geriye doğru yayılarak, her bir ağırlığın toplam hataya olan etkisi kısmi türev ve zincir kuralı kullanılarak hesaplanır. Şekil 2.4 ’de gösterilen iki katmanlı sinir ağında X, girdi matrisini, W1 , ilk gizli katman ağırlık matrisini, b1 , ilk gizli katman için yanlılık vektörünü, W2 , ikinci gizli katman ağırlık matrisini, b2 , bu katman için yanlılık vektörünü, ŷ ise çıktı değerlerini göstermektedir. Şekil 2.4’deki n0 özellik sayısını, n1 ilk gizli katman nöron sayısını, n2 ikinci gizli katman nöron sayısını, m ise örnek sayısını ifade etmektedir. Mevcut parametreler için maliyetin, J(W1 , b1, W2 , b2), sinir ağında geriye doğru yayılması, ağırlık ve yanlılık değerlerinin hata üzerindeki etkilerinin hesaplanması gerekmektedir. Bunun için, toplam hatanın ağdaki parametrelerin her birine göre kısmi değişim oranı (∂J/∂W2 , ∂J/∂b2, ∂J/∂W1 , ∂J/∂b1) zincir türev kuralı ile bulunur. Her hangi bir k katmanında gradyanı hesaplanan ağırlıklar belirlenen bir öğrenme oranı, α, ile güncellenir (denklem 2.7).

2.7)

Tüm girdilerin kullanılmasının ardından toplam hata belirlenen bir değerin altına ulaşana kadar toplam hatanın hesaplanması, mevcut parametrelerin hata üzerindeki etkisinin hesaplanması ve parametrelerin güncellenmesi işlemi tekrar ettirilir. Bazen belirlenen sayıda iterasyonun tamamlanması veya CPU zamanı da durdurma kriteri olarak verilebilir.

# **3. KONVOLÜSYONEL SİNİR AĞLARI**

Konvolüsyonel sinir ağları (KSA) [22, 23], görüntü sınıflandırma, nesne konumlandırma (localization), nesne algılama gibi bilgisayarlı görü çalışmalarında çokça kullanılmakta olan bir derin öğrenme mimarisidir. KSA, bilgisayarlı görü çalışmaları için hazırlanan birçok hazır veri setinde başarılı ve rekabetçi sonuçlar vermektedir [12, 23]. KSA, kendisini oluşturan katmanların en az birinde, filtrelerin girdiler üzerinde dolaşması (konvolüsyon) işleminin uygulandığı ve sonucunda özellik çıkarımının gerçekleştirildiği derin sinir ağlarıdır. KSA, anlatılan bu özelliği (konvolüsyon) ile diğer YSA’lardan ayrılır. KSA’lar bir tane girdi katmanı, çok sayıda konvolüsyon ve ortaklama katmanları ile tam bağlantılı katmandan oluşur.

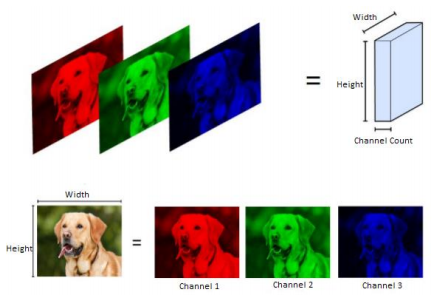
## Girdi Katmanı

Konvolüsyonel sinir ağının ilk katmanını oluşturur. Sinir ağının bir çıktı elde edebilmesi için gerekli olan görüntü girdi olarak verilir. Verilen girdinin bilgisayar tarafından anlaşılabilmesi için resmin pikselleri sayısal ifadelere dönüştürülmelidir. Verilen girdinin renkli bir resim olduğunu kabul edersek her bir piksel 0-255 arasında sayısal değerler ile ifade edilir.

Girdinin her bir pikselini temsil eden sayısal değerler matrislerde saklanır. Görüntünün genişliği, yüksekliği, renkli olup olmadığı gibi özellikler, görüntülerin saklandığı matrisin boyutlarını değiştirmektedir. Örneğin: 28x28 piksel boyutunda gri tonlamalı bir resim, girdi için 28x28x1 boyutunda bir matris yeterlidir. Vektör haline getirirsek 28x28x1 = 784 elemanlı bir vektör ile bu girdi temsil edilebilir. Fakat aynı boyutlardaki resmin renkli olduğunu düşünürsek 28x28x3 boyutunda bir matris kullanmamız gerekir. Yine bu matrisi vektör haline getirmek istersek 28x28x3 = 2352 elemanlı bir vektör elde etmiş oluruz. Renkli olan girdiler direkt olarak 3 boyutta temsil edilmektedir. Her bir boyut, kanal (channel) olarak isimlendirilir. Bunun nedeni 17 her bir girdinin RGB (Red, Green, Blue) kanallarının olmasındandır. Yani renkli bir girdinin her bir pikseli “Red” kanalında ayrı, “Blue” kanalında ayrı, “Green” kanalında ayrı bir sayısal değer ile ifade edilmektedir (Şekil 3.2). Bu yüzden seçilen girdi boyutları sinir ağının hızı ve çıktıların doğruluğu için çok önemlidir. Büyük girdi boyutları, çok fazla bellek kullanımına neden olup, ağın eğitilmesi sırasında yapılacak hesaplamaların yavaşlamasına neden olur. Fakat çıktıların doğru bir şekilde tahmin edilebilmesi için daha fazla özellik çıkarılabileceğinden etkili sonuçlar verebilir. Aynı şekilde küçük girdi boyutları, daha az bellek kullanımı sağlayıp, ağın eğitilmesi için yapılan hesaplamaları hızlandırabilir [2]. Fakat tahmin edilen çıktıların doğruluk oranı istenilen kadar başarılı olmayabilir. Bu yüzden sinir ağının başarısı ve hesaplama maliyeti gibi kriterler göz önünde bulundurularak doğru bir girdi boyutu seçilmelidir.

## Konvolüsyon Katmanı

Bu katman KSA’ların en önemli katmanlarından birini oluşturur. Bir önceki katmandan gelen girdiler üzerinde seçilen filtreler (kernel, receptive field size) uygulanarak yeni özellik kümeleri oluşturulur. Filtre boyutları değişkenlik gösterebilir ve matris yapısında tutulur (2x2, 3x3, 5x5 vb). Seçilen filtreler (ağırlık matrisleri) ile verilen girdi üzerinde dolaşılarak farklı özellikler elde edilmeye çalışılır. Girdiye uygulanan konvolüsyon filtreleri sonucunda her biri farklı özelliği temsil etmeye çalışan özellik haritaları elde edilir.

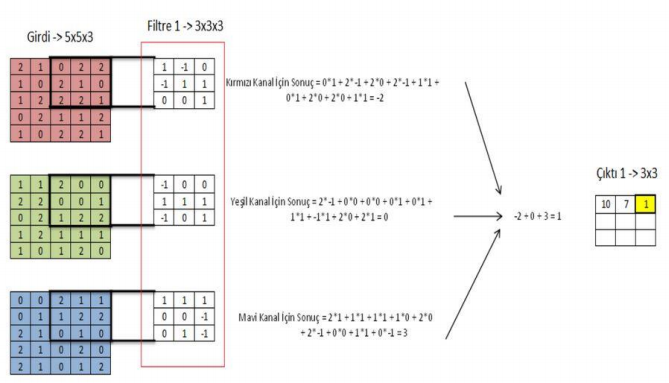


**Şekil 3.1** Renkli girdiler için kanal, genişlik ve yükseklik kavramlarının gösterimi

Girdiler üzerine uygulanacak filtrelerin nasıl seçileceği ve boyutlarının ne olacağı çok önemli bir noktadır. Filtrelerin genişlik ve yükseklik değerlerinin çok yüksek seçilmesi, girdi boyutunun çok hızlı bir şekilde küçülmesine neden olabilir. Düşük girdi boyutları sinir ağındaki hesaplama hızını arttırırken, girdi üzerinden elde edilebilecek ayırt edici özelliklerin kaybedilmesine sebep olabilir. Konvolüsyonel sinir ağlarındaki filtreler, sinir ağlarındaki ağırlık matrislerine denk olarak düşünülebilir. 18 Yapay sinir ağlarında olduğu gibi konvolüsyonel sinir ağlarında da ağırlıklar yani filtreler hesaplanan hataya göre belirlenen çevrimlerde güncellenir. Güncellenen filtre değerleri ile öğrenme işlemi devam ettirilmiş olur. Güncellenen filtre ağırlıkları, girdi üzerinde hangi özelliklerin daha önemli olduğunu belirtir. Öğrenilen bu bilgiler ilerleyen katmanlarda ağın geriye kalan kısımları ile paylaşılır (weight sharing). Güncellenen ağırlıklar sonucunda oluşan filtreler girdi üzerinde yeni özelliklerin keşfedilmesini sağlar. Konvolüsyon katmanında önemli olan bazı kavramlar vardır ve bu kavramlar şunlardır:

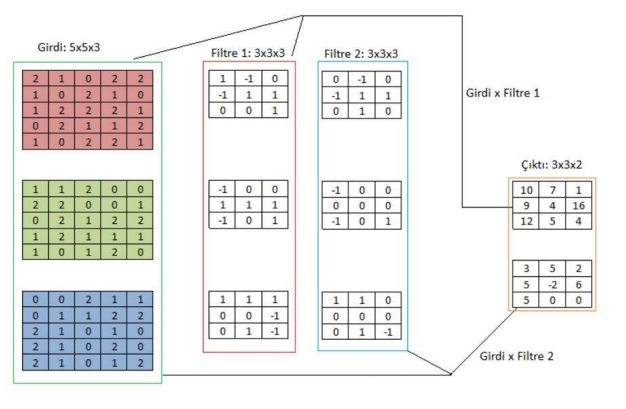
* **Filtre boyutu:** Girdi üzerine uygulanacak filtrenin yükseklik ve genişlik değerlerini temsil eder. Genelde kare matris olarak seçilir.
* **Aralık Değeri (Stride):** Filtre matrisinin, girdi üzerinde dolaşırken kaç adım kaydırılacağını belirler.
* **Dış Boşluk Sayısı (Padding):** Bu değer girdi matrisinin dışına eklenecek olan 0’lardan oluşan değer sayısını belirtir. Örneğin: 3x3 boyutunda bir girdiye padding = 1 uygulanırsa filtrenin yeni boyutu 5x5 olur. Bu özellik girdi boyutu ile çıktı boyutunun aynı kalmasının istendiği durumlarda kullanılır.
* **Özellik Haritaları (Feature Maps):** Girdi üzerine filtreler uygulandıktan sonra elde edilen özellik haritalarının sayısını belirtir.
* **Kanal Sayısı:** Girdi ve filtrelerin kaç kanaldan oluştuğunu temsil eder. Girdinin kanal sayısı ile girdiye uygulanacak olan filtrelerin kanal sayısı birbirine eşit olmalıdır. Örneğin: Girdi, 28x28x3 boyutunda ise yani 3 kanala sahip ise filtre ?x?x3 boyutunda olmalıdır (?: opsiyonel değerleri temsil eder, 3x3, 5x5, 7x7 vb).

Konvolüsyon işleminde yeni çıktılar, noktasal çarpım ve toplama işlemleri ile elde edilir. Konvolüsyon işleminde, belirlenen aralık hiper-parametresine (stride) göre konvolüsyon filtresi girdi üzerinde gezinir. Bu işlem sırasında, konvolüsyon filtresinin bulunduğu yere karşılık gelen alandaki girdi ağırlıkları ile konvolüsyon filtresinin ağırlıkları çarpılıp, toplanır. Bu işlem girdi boyunca her bir kanal için devam eder. Her bir kanal için elde edilen çarpımların toplamı alınarak, çıktı verisinde karşılık gelen noktaya bu toplam değer yazılır. Girdi üzerindeki her bir katman için farklı filtre 19 katsayıları seçilebilir. Örneğin: Konvolüsyon işlemi için kullanılacak olan girdinin genişliği 640 piksel, çıktının genişliği ise 638 piksel olsun. Her iki görüntü yüksekliği de 480 piksel olsun. Girdi görüntüsünden çıktıya dönüşüm gerçekleştirilmesi için konvolüsyon işlemi uygulanırsa; 638 x 480 x 3 = 918.720 kayan noktalı işlem gerekir. Aynı dönüşüm ağırlıkların tamamının birbirine bağlı olduğu bir sistemde matris çarpımı ile 94 milyardan (640 x 480 x 638 x 480) fazla hesaplama gerektirir. Bu durum konvolüsyon işlemini çok daha verimli hale getirir.



**Şekil 3.2** Girdi için 3 kanaldan oluşan filtrenin uygulanması (aralık = 1, dış boşluk = 0)

Denklem 3.1'e bakıldığında girdi matrisi ve ağırlık matrislerinin çarpılmasıyla yeni oluşan özellik haritalarının nasıl hesaplandığı görülmektedir. Bu notasyonda, W[i-1], i1. Katmanda bulunan her bir filtre için oluşturulan ağırlıkların saklandığı çok boyutlu bir matristir W[i-1] = [W1 [i-1] , W2 [i-1], ... , Wk [i-1]]. Z [i-1]matrisinde ise filtre ağırlık matrisi ile girdi ağırlık matrislerinin çarpılıp, önyargı vektörü ile toplanması sonucunda elde edilen yeni özellik haritaları saklanır (Z [i-1] = [Z1 [i-1],Z2 [i-1], ... , Zk [i-1]]). Bu notasyonda k değeri kernel, filtre sayılarını belirtmektedir. Verilen girdiler için konvolüsyon işleminin nasıl uygulandığı Şekil 3.2 ve Şekil 3.3 ’de gösterilmiştir.

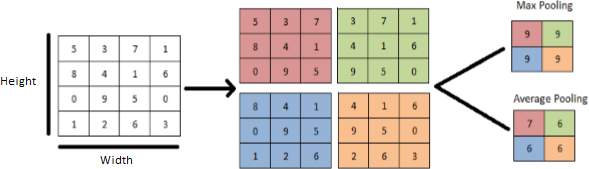
3.1)

**Şekil 3.3** Birden fazla filtre için konvolüsyon işlemin uygulanması ve çıktılar (aralık = 1, dış boşluk = 0)

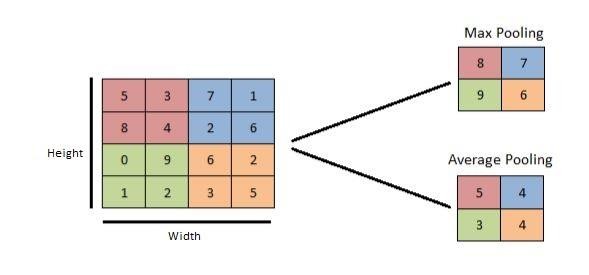
## Ortaklama Katmanı

Ortaklama (pooling) işlemi ya da bazı örneklerde alt örnekleme (subsampling) olarak geçen yöntem konvolüsyonel sinir ağlarında girdi ve özelliklerin boyutunun düşürülmesi için kullanılır. Girdi, ortaklama işlemine tabi tutulduğunda belirli bir matematiksel formüle göre yükseklik ve genişlik değerleri azalır. Özellik düşürme işlemi, özellik çıkarımı (öğrenimi) işlemleri sonucunda elde edilen özelliklerin kaybına neden olsa da KSA’nın eğitim süresini azaltır. Ortaklama işleminin en ilginç yanı öğrenilecek, elde edilen hataya göre güncellenecek hiper-parametrelerin olmayışıdır. Ortaklama katmanı, filtre boyutu, ortaklama yöntemi ve aralık değeri hiper-parametrelerine sahiptir. Fakat bu hiper-parametreler ağın optimizasyonuna dahil edilmez.

Ortaklama işleminin uygulanması için farklı yöntemler seçilebilir. Bu yöntemler ortaklama tipi, ortaklama yöntemi olarak adlandırılabilir ve KSA optimizasyonunda ortaklama katmanında seçilmesi gereken bir hiper-parametredir. Maksimum ortaklama (max pooling) ve ortalama ortaklama (average pooling) yöntemleri KSA’da sıklıkla kullanılmaktadır. Konvolüsyon katmanında olduğu gibi seçilen ortaklama filtreleri, girdiler üzerinde dolaşarak çıktıları elde eder. Ortalama ortaklama işleminde, seçilen aralık hiper-parametresine (stride) göre filtre girdi üzerinde gezinir. Bu işlem sırasında, girdi üzerinde ortaklama filtresinin bulunduğu yere karşılık gelen alandaki değerlerin ortalaması hesaplanıp, çıktı üzerinde karşılık gelen bölgeye yerleştirilir. Bu işlem girdi boyunca devam eder. Aynı işlem maksimum ortaklama işlemi için de geçerlidir. Fakat bu sefer elemanların ortalaması yerine girdide filtrenin bulunduğu 21 alana karşılık gelen kısımdaki maksimum değer çıktı olarak verilir. Farklı aralık (stride) değerleri için girdilere “Max ve Average Pooling” işlemlerinin uygulanması Şekil 3.4 ve 3.5 ’da gösterilmiştir.



**Şekil 3.4** 3x3 Filtre boyutu ve 1 aralık (stride) değeri için maksimum ve ortalama ortaklama çıktıları



**Şekil 3.5** 2x2 Filtre boyutu ve 2 adım (stride) değeri için maksimum ve ortalama ortaklama çıktıları

## Tam Bağlantılı Katman

Konvolüsyonel sinir ağlarında özellik çıkarımı işleminden sonra, elde edilen özelliklerin sınıflandırılması için kullanılan son katmanları içerir. Tam bağlantılı katmandan önce, verilen girdi üzerinden özelliklerin çıkarılması, öğrenilmesi için konvolüsyon, ortaklama, normalizasyon ve aktivasyon işlemleri ağın yapısına bağlı olarak birkaç defa tekrarlanır. Elde edilen özellikler sonucunda artık bir tahmin işleminin gerçekleştirilmesi gerekmektedir. Tam bağlantılı katman kendinden önce gelen diğer katmanlarla tamamen bağlıdır. Bundan sonra kullanılacak olan yapı çıkartılan özelliklerin ağırlıklandırılması ve bir çıktının tahmin edilmesi işlemini gerçekleştirecektir. İstenilen doğruluk değeri elde edilene veya sonlandırma kriteri (iterasyon sayısı) sağlanana kadar ağırlıklar hesaplanan hataya göre güncellenir. Tam bağlantılı katmanın yapısı, yapay sinir ağlarının yapısı ile benzerlik göstermektedir. Bir giriş katmanı, isteğe bağlı olarak bir ya da birden fazla gizli katman ve son olarak çıktı katmanından oluşur.

Tam bağlantılı katmana gelene kadar elde edilen özelliklerin boyutunun 16x16x128 olduğunu varsayalım. Yani 16x16 boyunda 128 tane özellik elde edilmiş olsun. Bu matrisi bir vektör haline çevirmemiz gerekir. Bu yüzden elde edilen matrisi 16x16x128=32.768x1’lik bir vektöre çevirerek giriş katmanına veririz. Bu işlemin adına “Flattening” denir. Aynı zamanda giriş katmanı ile çıktı katmanı arasına bir tane gizli katman eklemek istersek ve bu katmanın 512x1 boyutunda bir vektör olduğunu varsayarsak toplamda 32.768x512 boyutunda bir ağırlık matrisi elde edilmiş olur [2]. Son olarak ta tahmin edilecek sınıf boyutunda bir çıktı katmanı eklenir.

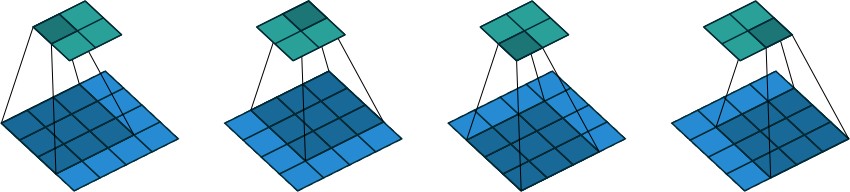
Görüldüğü gibi elde edilen ağın toplam parametre sayısı yükseldiğinden, eğitilmesi için yüksek hesaplama gücü gerekmektedir. Bu yüzden tamamen bağlı katmana gelene kadar olan bölümde, yani özellik çıkarımının gerçekleştiği katmanlarda giriş katmanından ileriye doğru giderken kanal sayısı arttırılmalı, yükseklik x genişlik sayısı azaltılmalıdır. Yani girdinin yükseklik ve genişlik değerleri küçültülmeli, fakat özellik haritası sayısı arttırılmalıdır.

## Konvolüsyon (Convolution) Aritmetiği

Bölüm 3.2 ’de konvolüsyon işleminin ne olduğundan ve nasıl çalıştığından bahsedildi. Konvolüsyon işlemi yapıldıktan sonra elde edilen çıktıların boyutu neye göre belirlenir, konvolüsyon katmanında aralık (stride) ve dış boşluk (padding) değerleri ne için kullanılır ve konvolüsyon işleminden sonra elde edilen çıktının boyutunun değişmesini istemediğimiz durumlarda neler yapılması gerektiği bu kısımda anlatılacaktır.

## Sıfır dış boşluk (padding) ve bir adım değeri

Şekil 3.6’de dış boşluk (padding) değerinin sıfır, adım (stride) değerinin 1 olduğu durum gösterilmiştir. Bu durumda, girdiye konvolüsyon işlemi uygulandıktan sonra elde edilecek çıktının boyutu denklem 3.2’de [24] gösterilen formüle göre hesaplanır (ç: çıktı boyutu, g: girdi boyutu, f: filtre boyutu).

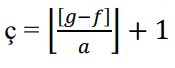
3.2) Ç = (G – F) + 1

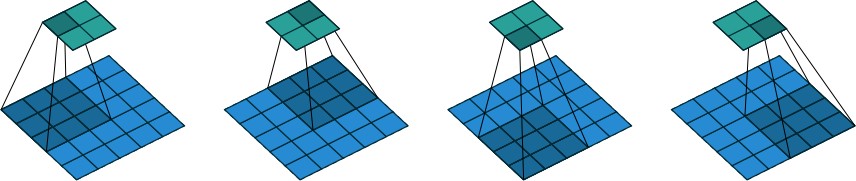
**Şekil 3.6** 4x4 boyutunda bir girdi 3x3’lük bir filtre ile konvolüsyon işlemine sokuluyor (dış boşluk = 0, stride = 1, çıktı boyutu = 2x2)

Şekil 3.6’de gösterilen örnek için elde edilecek çıktı Denklem 3.2’ye göre hesaplanır: Çıktı Boyutu = (4 – 3) + 1 = 2

## Sıfır dış boşluk (padding) ve birden farklı adım değeri

Dış boşluk (padding) değerini sıfır kabul edip, aralık sayısının değişken olduğunu kabul edersek verilen girdi için elde edilecek çıktı değerini Denklem 3.3’te [24] gösterilen formüle göre hesaplayabiliriz (ç: çıktı boyutu, g: girdi boyutu, f: filtre boyutu, a: aralık sayısı).

3.3)

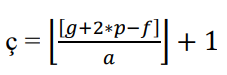
**Şekil 3.7** 5x5 boyutunda bir girdi 3x3’lük bir filtre ile konvolüsyon işlemine sokuluyor (dış boşluk = 0, stride = 2, çıktı boyutu = 2x2)

Şekil 3.7’de gösterilen örnek için elde edilecek çıktı Denklem 3.3’e göre hesaplanır:

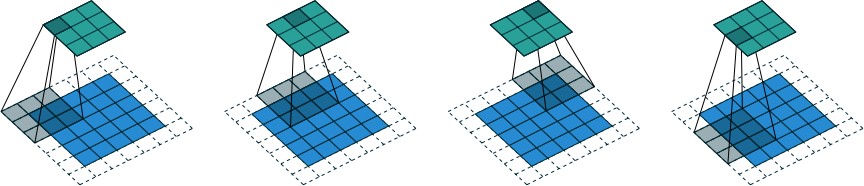


Çıktı Boyutu =

## Sıfırdan farklı dış boşluk ve aralık sayısı

Girdiye uygulanan konvolüsyon işleminden sonra elde edilecek olan çıktı boyutunun hesaplanabileceği en genel formüldür. Konvolüsyon işleminin ardından elde edilen çıktı boyutunun hesaplandığı genel formül denklem 3.4’te [24] verilmiştir (ç: çıktı boyutu, g:girdi boyutu, p: dış boşluk sayısı, f: filtre boyutu, a: aralık sayısı).

3.4)



**Şekil 3.8** 5x5 boyutunda bir girdiye 1x1’lik kenar ekleniyor ve 3x3’lük bir filtre ile konvolüsyon işlemine sokuluyor. (dış boşluk = 1, stride = 2, çıktı boyutu = 3x3)

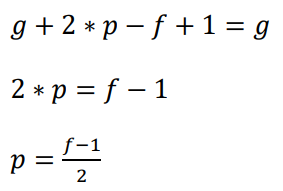
Şekil 3.9’da gösterilen örnek için elde edilecek çıktı Denklem 3.4’e göre hesaplanır:

Çıktı Boyutu =

Görüldüğü gibi konvolüsyon işleminde farklı parametre değerleri için farklı çıktı boyutları elde edilmektedir. Elde edilecek olan çıktının mimari kurulurken hesaplanması, bilinmesi büyük bir önem arz etmektedir. Verilen girdi için çıktı katmanında elde edilecek boyutun hesaplanması için denklem 3.4 kullanılabilir. Bu denklem elde edilecek çıktının hesaplanması için kullanılan formüllerin en genel halidir. Denklem 3.4, çıktı boyutunu doğrudan etkileyen, girdi boyutu, filtre boyutu, aralık sayısı ve iç boşluk sayısı gibi bütün parametreleri içermektedir.

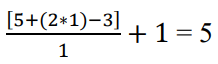
## Girdi boyutunun korunması

Bazı durumlarda konvolüsyon işleminden elde edilen çıktının girdi ile aynı boyutta olmasını isteriz. Böyle durumlarda dış boşluk (padding) parametresini kullanırız. Bu duruma “same padding” denir. Girdi ile çıktının aynı boyutta olması için kullanılacak olan iç boşluk parametresi belirli bir denkleme göre hesaplanır. Bu denklem aralık sayısının bir olduğu durumlar için doğru sonucu vermektedir. Çünkü filtre boyutu ve aralık sayısı gibi değişkenlerin farklı olması sonucu değiştirebilir (g: girdi boyutu, p: dış boşluk sayısı, f: filtre boyutu).

3.5)

Aralık sayısının 1 olduğu durumlarda dış boşluk (padding) sayısı denklem 3.5’e göre hesaplanır. Örneğin: 5x5 boyutundaki bir girdiye, aralık sayısı (stride) 1 iken, 3x3 boyutunda bir filtre uygulandığında tekrar 5x5 boyutunda bir çıktı elde etmek istersek kullanmamız gereken dış boşluk sayısı Denklem 3.5’e göre hesaplanabilir:



Bulunan “p” değerini Denklem 3.4’te yerine koyarsak:

Çıktı Boyutu =

sonucu elde edilir. Görüldüğü gibi girdi boyutu ile çıktı boyutu aynı değerdedir.

## Ortaklama (Pooling) Aritmetiği

Havuzlama işleminde iç boşluk(padding) değeri kullanılmadığından sadece filtre boyutu, girdi boyutu ve adım sayısı değerleri kullanılır. Konvolüsyon işlemindeki genel formülden (Denklem 3.4) tek farkı dış boşluk (padding) değerinin olmayışıdır. Bu nedenle ortaklama katmanında kullanılan girdi boyutu, filtre boyutu ve aralık sayısı değerleri için elde edilecek çıktının boyutu denklem 3.6’ya göre hesaplanır. Bu denklem 3.3 ile aynıdır (g: girdi boyutu, f: filtre boyutu, a: adım sayısı).

3.6)

Örneğin: 32x32 boyutundaki bir girdi için, aralık değeri (stride) 2 olan 2x2 ortaklama boyutunda (pooling size) bir filtre uygulanırsa, elde edilecek çıktının boyutu denklem 3.6’ya göre hesaplanır:

Çıktı Boyutu =

## Fashion-MNIST veri seti

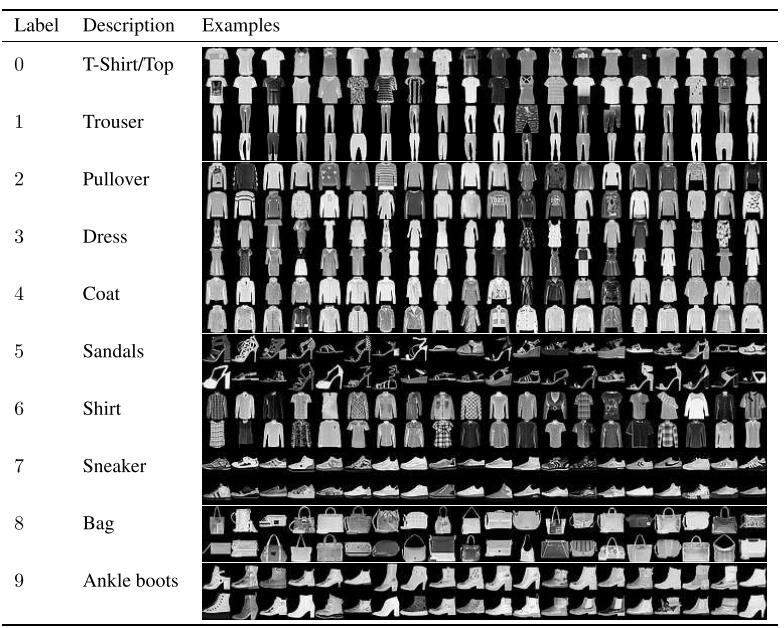
Fashion-MNIST veri seti 28x28 boyutunda, gri tonlamalı 70.000 adet moda ürününden oluşan 10 sınıflık bir veri setidir. Her kategori için 7.000 resim vardır. Eğitim için 60.000, test için 10.000 resim bulunmaktadır. MNIST el yazısı veri setinin günümüzde kullanılan konvolüsyonel sinir ağları için kolay tahmin edilebilir olması ve yüksek doğruluk oranları vermesi yeni veri seti ihtiyacı doğurmuştur. Bu ihtiyacı karşılamak için Zalando giyim markasının web sitesi kullanılarak Fashion-MNIST veri seti oluşturulmuştur [31].

Diğer veri setlerine göre çok daha yakın bir tarihte oluşturulan ve 2017 yılında herkese açık bir şekilde yayınlanan Fashion-MNIST [31] veri seti konvolüsyonel sinir ağı çalışmalarında yeni yeni kullanılmaya başlanmıştır. Veri setine ait özellikler Tablo 3.3’de, örnek görüntüler ise şekil 3.20’de gösterilmiştir.

|  |  |
| --- | --- |
|  | **Fashion MNIST Veri Seti Özellikleri** |
| **Tanım:** | Zalando giyim markasının web sitesi kullanılarak oluşturulan veri setidir. 10 farklı sınıf için moda ürünlerine ait görüntüleri  içermektedir. |
| **Boyut:** | 28x28 |
| **Eğitim Veri Sayısı:** | 60.000 |
| **Test Veri Sayısı:** | 10.000 |
| **Tip:** | Gri Tonlamalı (Tek kanal) |
| **Sınıf Sayısı:** | 10 |

Fashion-MNIST veri seti 10 sınıf içermektedir. Bu sınıflar şunlardır:

|  |  |
| --- | --- |
| * T-Shirt | * Sandalet |
| * Pantolon | * Gömlek |
| * Süveter | * Spor ayakkabı |
| * Elbise | * Çanta |
| * Ceket | * Ayak bileği çizmeler |



**Şekil 3.16** Fashion-MNIST veri seti sınıflar ve örnek görüntüler

# **4. UYGULAMA**



## Veri Seti

import tensorflow as tf

from tensorflow import keras

from keras.utils import to\_categorical

from keras.preprocessing.image import ImageDataGenerator

from keras.models import Sequential

from keras.layers import Dense, Dropout, Activation, Flatten

from keras.layers.normalization import BatchNormalization

from keras.layers import Conv2D, MaxPooling2D

import numpy as np

from keras.utils import np\_utils

from keras.callbacks import TensorBoard

import pickle

import time

from keras.datasets import fashion\_mnist

from keras.optimizers import SGD

(trainX, trainY), (testX, testY) = fashion\_mnist.load\_data()

trainX = trainX.reshape((trainX.shape[0], 28, 28, 1))

testX = testX.reshape((testX.shape[0], 28, 28, 1))

trainY = to\_categorical(trainY)

testY = to\_categorical(testY)

chanDim = 1

## CNN Modeli

model = Sequential()

# convert from integers to floats

train\_norm = trainX.astype('float32')

test\_norm = testX.astype('float32')

# normalize to range 0-1

train\_norm = train\_norm / 255.0

test\_norm = test\_norm / 255.0

model.add(Conv2D(32, (3, 3), padding="same",input\_shape=(28, 28, 1)))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(32, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# second CONV => RELU => CONV => RELU => POOL layer set

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(Conv2D(64, (3, 3), padding="same"))

model.add(Activation("relu"))

model.add(BatchNormalization(axis=chanDim))

model.add(MaxPooling2D(pool\_size=(2, 2)))

model.add(Dropout(0.25))

# first (and only) set of FC => RELU layers

model.add(Flatten())

model.add(Dense(512))

model.add(Activation("relu"))

model.add(BatchNormalization())

model.add(Dropout(0.5))

# softmax classifier

model.add(Dense(10))

model.add(Activation("softmax"))

NUM\_EPOCHS = 25

INIT\_LR = 1e-2

BS = 32

opt = SGD(lr=INIT\_LR, momentum=0.9, decay=INIT\_LR / NUM\_EPOCHS)

model.compile(loss="categorical\_crossentropy", optimizer=opt, metrics=["accuracy"])

model.fit(trainX, trainY, batch\_size=32, epochs=10, validation\_data=(testX, testY))

model.save('final\_model.h8')

>>>>>

Train on 60000 samples, validate on 10000 samples

Epoch 1/10

60000/60000 [==============================] - 381s 6ms/step - loss: 0.6519 - acc: 0.7815 - val\_loss: 0.4264 - val\_acc: 0.8477

Epoch 2/10

60000/60000 [==============================] - 382s 6ms/step - loss: 0.3821 - acc: 0.8636 - val\_loss: 0.3111 - val\_acc: 0.8841

Epoch 3/10

60000/60000 [==============================] - 380s 6ms/step - loss: 0.3263 - acc: 0.8844 - val\_loss: 0.2703 - val\_acc: 0.9026

Epoch 4/10

60000/60000 [==============================] - 380s 6ms/step - loss: 0.2918 - acc: 0.8948 - val\_loss: 0.2539 - val\_acc: 0.9065

Epoch 5/10

60000/60000 [==============================] - 382s 6ms/step - loss: 0.2715 - acc: 0.9013 - val\_loss: 0.2437 - val\_acc: 0.9102

Epoch 6/10

60000/60000 [==============================] - 375s 6ms/step - loss: 0.2580 - acc: 0.9058 - val\_loss: 0.2299 - val\_acc: 0.9165

Epoch 7/10

60000/60000 [==============================] - 373s 6ms/step - loss: 0.2473 - acc: 0.9106 - val\_loss: 0.2233 - val\_acc: 0.9190

Epoch 8/10

60000/60000 [==============================] - 372s 6ms/step - loss: 0.2346 - acc: 0.9142 - val\_loss: 0.2260 - val\_acc: 0.9148

Epoch 9/10

60000/60000 [==============================] - 370s 6ms/step - loss: 0.2276 - acc: 0.9171 - val\_loss: 0.2151 - val\_acc: 0.9224

Epoch 10/10

60000/60000 [==============================] - 375s 6ms/step - loss: 0.2241 - acc: 0.9179 - val\_loss: 0.2101 - val\_acc: 0.9224

## Tahmin

from keras.preprocessing.image import load\_img

from keras.preprocessing.image import img\_to\_array

from keras.models import load\_model

from keras.datasets import fashion\_mnist

#model = load\_model('final\_model.h8')

# load and prepare the image

def load\_image(filename):

    # load the image

    img = load\_img(filename, grayscale=True, target\_size=(28, 28))

    # convert to array

    img = img\_to\_array(img)

    # reshape into a single sample with 1 channel

    img = img.reshape(1, 28, 28, 1)

    # prepare pixel data

    img = img.astype('float32')

    img = img / 255.0

    return img

# load an image and predict the class

def run\_example():

    # load the image

    img = load\_image('1.png')

    # load model

    model = load\_model('final\_model.h8')

    # predict the class

    result = model.predict\_classes(img)

    print(result[0])

# entry point, run the example

run\_example()

from keras.preprocessing.image import load\_img

from keras.preprocessing.image import img\_to\_array

from keras.models import load\_model

from keras.datasets import fashion\_mnist

#model = load\_model('final\_model.h5')

# load and prepare the image

def load\_image(filename):

    # load the image

    img = load\_img(filename, grayscale=True, target\_size=(28, 28))

    # convert to array

    img = img\_to\_array(img)

    # reshape into a single sample with 1 channel

    img = img.reshape(1, 28, 28, 1)

    # prepare pixel data

    img = img.astype('float32')

    img = img / 255.0

    return img

# load an image and predict the class

def run\_example():

    # load the image

    img = load\_image('1.png')

    # load model

    model = load\_model('final\_model.h8')

    # predict the class

    result = model.predict\_classes(img)

    print(result[0])

# entry point, run the example

run\_example()

# **DEĞERLENDİRME VE ÖNERİLER**

Günümüzde konvolüsyonel sinir ağları kullanılarak yapılan bilgisayarlı görü çalışmalarında çok başarılı sonuçlar elde edilse de kullanılan yöntem hala birçok zorluk içermektedir. Bu zorluklardan biri gittikçe derinleşen sinir ağlarının gerektirdiği yüksek hesaplama maliyetidir. Aynı zamanda konvolüsyonel sinir ağları kurulurken ağ modelindeki katman sayısı, filtre boyutu, özellik haritası sayısı, öğrenme oranı ve daha birçok hiper-parametrenin en başarılı sonucu verecek şekilde seçilmesi gerekmektedir. Bu seçim işlemi, yani optimizasyon işlemi hesaplama maliyeti yüksek bir işlem olduğundan ızgara arama yöntemi gibi oluşabilecek bütün hiper-parametre kombinasyolarının eğitilip, test edildiği yöntemler uygulanabilir doğruluk oranlarının makul sürelerde elde edildiği üst-sezgisel yöntemler gereklidir kullanılmıştır. KSA hiper-parametre optimizasyonu çalışmaları için GA ve PSO üst-sezgisel yöntemleri sıklıkla kullanılmış ve özellikle GA ile hiper-parametre optimizasyonu gerçekleştirilen çalışmalarda başarılı sonuçlar elde edilmiştir (Tablo A). Üst-sezgisel yöntemler kullanılarak yapılan çalışmalarda göze çarpan en büyük eksiklik, bu algoritmaların parametre seçimi için sistematik bir optimizasyon çalışması gerçekleştirilmemiş olmasıdır. Örneğin: GA için başlangıçta belirlenen popülasyon büyüklüğü, çaprazlama ve mutasyon olasılığı gibi parametreler kullanılmış, farklı parametrelerle algoritmanın nasıl sonuçlar verdiği test edilmemiştir. Aynı zamanda, KSA hiper-parametre optimizasyonu çalışmalarında optimize edilmeye çalışılan parametrelerde bir bütünlük görülmemektedir. Bu çalışmalarda genel olarak parametreler kısmi olarak ele alınmış; bir çalışmadan diğerine geçildiğinde çok farklı parametre alt kümeleri optimize edilmeye çalışılmıştır. Bazı çalışmalarda sabit mimari kullanılırken bazı çalışmalarda dinamik olarak genişleyen bir mimari kullanılmıştır. Hatta bazı çalışmalarda optimize edildiği söylenen parametreler için nispeten dar değer aralıkları seçilmiştir. Birçok çalışmada öğrenme oranı, yığın büyüklüğü ve optimizasyon metodu gibi önemli hiper-parametreler optimizasyon işlemine dahil edilmemiştir. Aynı zamanda elde edilen en iyi topolojiler için toplam parametre sayısı ve bu topolojiyi oluşturmak için gereken tüm hiper-parametre bilgileri eksiksiz olarak verilmemiştir. Çoğu çalışmada yapılan kabuller ile ilgili doyurucu sebepler belirtilmemiştir. Bu sebeplerden dolayı literatürde gerçekleştirilen çalışmalar ile bu tez çalışmasında önerilen yöntemlerin kapsamlı bir şekilde karşılaştırılması zorlaşmaktadır. KSA hiper-parametre optimizasyonu için optimizasyon yöntemi seçilirken dikkat edilmesi gereken önemli noktalardan biri optimize edilecek hiper-parametre kümesinin büyüklüğüdür. Daha dar, küçük hiper-parametre kümeleri için ızgara arama yöntemi seçilerek etkili sonuçlar elde edilebilirken, daha geniş, büyük hiper-parametre kümelerinde daha başarılı sonuçların elde edilebilmesi için rastgele arama, üst- sezgisel algoritmalar gibi yöntemler seçilebilir. GA, PSO, DE gibi üst-sezgisel yöntemler ile daha iyi doğruluk oranları elde edilirken, bu yöntemler kullanıldığında daha fazla hesaplama süresi ihtiyacı oluşmaktadır. Literatürdeki çalışmalar incelendiğinde daha az eğitim, hesaplama süreleri için TPE, SMAC gibi yöntemlerin önerildiği görülmüştür. Üst-sezgisel algoritmaların seçiminde, parametre ayarlaması zorluğundan dolayı daha az parametreye sahip üst-sezgisel yöntemler, daha çok parametreye sahip olan üst-sezgisel yöntemlere tercih edilebilir. Bunun yanında, implementasyon kolaylığı ve daha hızlı yakınsama süreleri, yani daha az sayıda iterasyonda iyi sonuçlar üretebilmesi de üst-sezgisel seçiminde önemli olabilmektedir. Örneğin: PSO, GA’ ya kıyasla daha az sayıda parametreye sahiptir, hızlı çalışmakta ve GA’ ya kıyasla rekabetçi sonuçlar vermektedir. Yüksek doğruluk oranı/ düşük hata oranı veren yöntemler arasından ise en hızlı çalışan yöntem çoğu zaman tercih edilen yöntem olabilir. Literatürde sıklıkla kullanılan üst-sezgisel yöntemlerin aksine bu çalışmada KSA hiper-parametre optimizasyonu için Mikrokanonikal Optimizasyon yöntemi kullanıldı. Benzetimli tavlama yönteminin farklı bir varyasyonu olan bu yöntemin, benzetimli tavlama yöntemine göre kendi içerisinde daha az parametreye sahip olmasının daha avantajlı olabileceğini düşündüm. Aynı zamanda sahip olduğum hesaplama gücü, donanım desteği sınırlı olduğundan popülasyon tabanlı yöntemleri tercih etmedim. Yapılan bu tez çalışması sonucunda elde edilen sonuçlara bakıldığında doğruluk oranı bakımından literatürdeki çalışmalarla (CIFAR10 veri seti hariç) rekabetçi sonuçlar elde edildi. Aynı zamanda düşük parametre sayılarından oluşan topolojiler ile hesaplama süresi bakımından başarılı sonuçlar elde edildi. Literatürde state-of-the art olarak kabul edilen mimarilerin birçoğundan hem doğruluk oranı hem de hesaplama süresi bakımından daha iyi sonuçlar elde edildi. Donanım desteği ve hesaplama gücünün yetersiz olması nedeniyle kısıtlanmak zorunda kalan topolojiler ve µO-Kısıtlı yönteminin, daha yüksek hesaplama gücü elde edildiğinde, daha fazla ve daha geniş topolojiye sahip çözümler üretileceğinden daha da iyi sonuçlar vereceğini düşünmekteyim. Hiper-parametre optimizasyonu çalışmalarında üretilen topolojiler çok iyi doğruluk değerlerine sahip olsalarda çok uzun eğitim sürelerine ihtiyaç duyabilmektedirler. Bu durum yüksek hesaplama zamanı yanında mobil cihazlar için test sırasında yüksek enerji tüketimine de neden olmaktadır. Bu nedenle, yapılan KSA hiper-parametre optimizasyonu çalışmalarının aksine, daha az parametre sayısına sahip yani çok daha az eğitim süresine, hesaplama zamanına ihtiyaç duyan, eğitim adımlarını hızlandıran ve kabul edilebilir seviyede doğruluk oranlarının elde edilmeye çalışıldığı çok amaçlı KSA hiper-parametre optimizasyonu çalışmalarının ileride daha çok çalışılacağını düşünmekteyim.

# **KAYNAKLAR**

1. McCulloch, W. S., Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.
2. ÜLKER, E. Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri. Gaziosmanpaşa Bilimsel Araştırma Dergisi, 6(3), 85-104.
3. Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. Psychological review, 65(6), 386.
4. Minsky M., Papert S. (1969), Perceptrons. MIT Press, Cambridge, MA
5. Rumelhart, D. E., Hinton, G. E., Williams, R. J. (1986). Learning representations by back-propagating errors. nature, 323(6088), 533.
6. Cortes, C., V. Vapnik. Support-vector networks. Machine learning, 20(3), (1995) 273-297.
7. DNI Instute. [Çevrimiçi] [Alıntı Tarihi: 28 05 2019] http://dni-institute.in/blogs/building-predictive-model-using-svm-and-r/
8. LeCun, Y., Bengio, Y. (1995). Convolutional networks for images, speech, and time series. The handbook of brain theory and neural networks, 3361(10), 1995.
9. Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. In Advances in neural information processing systems (pp. 1097-1105).
10. E. Alpaydin (2009). Introduction to machine learning. MIT press.
11. E. Öztemel (2003), Yapay Sinir Ağları. Istanbul: Papatya Yayincilik.
12. I. Goodfellow, Y. Bengio, A. Courville and Y. Bengio (2016), Deep learning (Vol. 1). Cambridge: MIT press
13. X. Glorot, B. Yoshua, Understanding the difficulty of training deep feedforward neural networks. Proceedings of the thirteenth international conference on artificial intelligence and statistics, (2010).
14. B. Karlik, A. V. Olgac, Performance analysis of various activation functions in generalized MLP architectures of neural networks. International Journal of Artificial Intelligence and Expert Systems, 1:4 (2011) 111-122.
15. K. He, X. Zhang, S. Ren, J. Sun, Delving deep into rectifiers: Surpassing human-level performance on imagenet classification. Proceedings of the IEEE international conference on computer vision, (2015).
16. M. Sinecen, B. Kaya, Ö. Yıldız, Artificial Neural Network Based Early Warning System For Aydin Province Towards Air Factors Which Primarily Affect Human Health. Gazi Üniversitesi Fen Bilimleri Dergisi Part C: Tasarım ve Teknoloji, 5:4 (2017) 121-131. DOI: 10.29109/http-gujsc-gazi-edu- tr.304938.
17. T. Yichuan, Deep learning using linear support vector machines. arXiv preprint arXiv:1306.0239, (2013).
18. Srivastava, N., Hinton, G., Krizhevsky, A., Sutskever, I., Salakhutdinov, R. Dropout: a simple way to prevent neural networks from overfitting. The Journal of Machine Learning Research, 15(1), (2014) 1929-1958.