KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ

BİTİRME TEZİ

PROJE (TEZ) BAŞLIĞI: Derin Öğrenme Tabanlı Nesne Sezme ve Tanıma

TEMATİK ALAN: Görüntü İşleme

PROJE YÜRÜTÜCÜSÜ: Mehmet Can AKAY

EĞİTİM KURUMU: Kocaeli Üniversitesi

DANIŞMAN: Prof. Dr. Yaşar BECERİKLİ

KOCAELİ Ocak 2019

ÖNSÖZ VE TEŞEKKÜR

Çalışmalarım boyunca değerli yardım ve katkılarıyla bana yön veren saygıdeğer Hocam Prof. Dr. Yaşar BECERİKLİ'ye teşekkürlerimi sunarım.Ayrıca çalışmam boyunca hiçbir türlü yardımı esirgemeyen araştırma görevlisi Fulya AKDENİZ hocama ve diğer emeği geçen herkese teşekkür ederim.

Ocak-2019 Mehmet Can AKAY

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

Derin Öğrenme Tabanlı Nesne Tanıma Ve Sezme

ÖZET

Çalışmanın konusu, bir resmin, derin öğrenme metodları kullanılarak bilgisayar tarafından belirlenip, tanınmasıdır. Bunun için öncellikle tanınmasını istediğimiz nesne veya kişiye ait resimlerden oluşan bir veritabanı oluşturulması gerekiyor.Bu kişi veya nesnenin bilgisayar tarafından tanınmasını sağlamak için çok sayıda ve net görüntünün yanında her olasılığın incelenebilmesi için net olmayan görüntüleride içermesi gerekmektedir.
Bu datasetimizi Tensorflow nesne tanıma uygulama programlama arayüzünün içine atıyoruz.Burada önceden eğitilmiş Tensorflow modelleri ile eğitilmek için.Bu model de derin öğrenme methodu olan konvolüsyonel sinir ağı kullanılmıştır.
Konvolüsyonel sinir ağında datasetimizdeki resimler konvolüsyon katmanından geçirilip önceden eğitilmiş modeller üzerinden tanınıp, tespit ediliyor.

Anahtar Kelimeler: Görüntü İşleme,Nesne Tanıma,Nesne Belirleme,Derin Öğrenme,Konvolüsyonel Sinir Ağı

Deep Learning Based Object Detection And Recognation

ABSTRACT

The subject of the study is to determine and recognize a picture by using computer by using deep learning methods. For this purpose, we need to create a database of images of the object or person we want to be recognized first. In order to enable this person or object to be recognized by the computer, it is necessary to include many and clear images as well as unclear images in order to examine each event.
We are putting this datasets into the Tensorflow object recognition application programming interface. To be trained with pre-trained tensorflow models. This model has been used convulsional neural network with deep learning method.
In the convulsive neural network, the images in our data are passed through the convolution layer and identified and detected on pre-trained models.

Key Words: Image Processing, Object Recognition, Object Detection, Deep Learning, Convolutional Neural Network

1.GİRİŞ

1.1 CoNN Çıkış Noktası Biyolojik Bağlantı

Konvolüsyonel sinir ağları, sinirbilim veya biyoloji ile ilgili olarak görsel korteksten biyolojik bir ilham alır. Görsel korteks, görsel alanın belirli bölgelerine duyarlı küçük hücreli bölgelere sahiptir. Bu fikir, 1962'de Hubel ve Wiesel tarafından yapılan büyüleyici bir deney ile genişletildi; burada beyindeki bazı bireysel nöronal hücrelerin sadece belirli bir yönelimin kenarlarında olduğunu(veya ateşlendiğini) gösterdiler. Örneğin, bazı nöronlar dikey kenarlara maruz kaldığında ve bazıları yatay veya diyagonal kenarlar gösterildiğinde ateşlenir. Hubel ve Wiesel, tüm bu nöronların bir sütun mimarisinde organize edildiğini ve birlikte görsel algı üretebildiklerini keşfetti. Özel görevleri olan bir sistemin içindeki özel bileşenler fikri(görsel korteksteki özel karakteristikler için nöronal hücreler), makinelerin de kullandığı ve CoNN'lerin ardındaki temeldir.

1.2 Yapay Sinir Ağlarının Genel Yapısı

Dış ortamdan veya diğer hücrelerden alınan girdiler, ağırlıklar yardımıyla hücreye bağlanır. Toplama fonksiyonu ile net girdi hesaplanır. Net girdinin aktivasyon fonksiyonundan geçirilmesiyle net çıktı hesaplanır. Bu işlem aynı zamanda bir hücrenin çıkışını verir. Her bağlantının bir ağırlık değeri vardır. YSA, kendisine örnekler gösterildikçe bu ağırlık değerlerini değiştirir. Hedef, ağa gösterilen örnekler için doğru çıkışları verecek ağırlıkları bulmaktır.

Örnekler çok sayıda nitelik çiftleri ile temsil edilmektedir. Öğrenilecek olan hedef fonksiyonu önceden tanımlanmış olan özellikler vektörü ile tanımlanabilir. Hedef fonksiyon ayrık değerli, reel değerli yada ayrık ve reel değerlerden oluşan vektör şeklinde olabilir. Eğitim verileri hata içerebilir. YSA öğrenme metotları eğitim verisindeki gürültülere dayanıklıdır. Uzun süren eğitim zamanları olabilir. Öğrenme süresi uzundur fakat, öğrenilen durumun yeni örneklerde test edilmesi yani değerlendirilmesi hızlıdır. Hedef fonksiyonu öğrenme yeteneğini anlamak önemli değildir. YSA tarafından öğrenilen ağırlıkların yorumlanması zordur.

Bilgi ağın bağlantı ağırlıkları ile temsil edilir. Ağın sahip olduğu ağırlık değerlerinin doğru olduğu ölçüde ağın performansı fazladır. Bilgi dağıtılmıştır, ağırlık değerlerinin bazıları kaybolsa dahi ağ çalışmasını sürdürebilir.

Derin Öğrenme Özellikleri Bölüm 2.1 de detaylı şekilde açıklanmıştır.

1.3. Yapay Sinir Ağları Kullanım Alanları

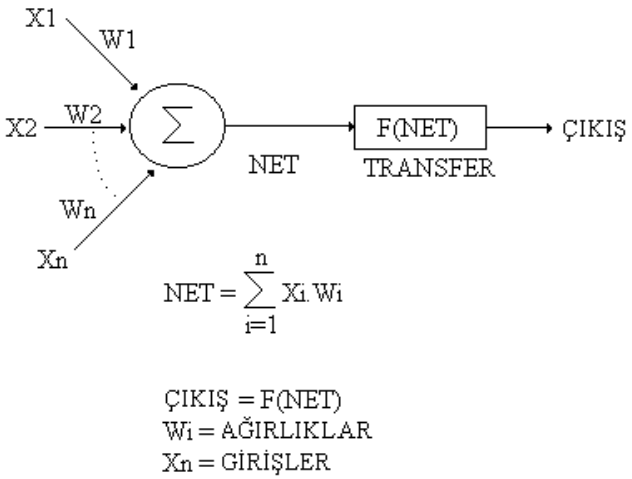
Geçmişten günümüze yapay sinir ağları birçok alanda ve uygulamalarda kullanılmaktadır. Bu uygulama alanlarına; üretim planlama, kan analizlerinin sınıflandırılması, beyin modellemesi çalışmaları, kalite kontrolü, parmak izi tanıma, otomatik araç denetimi, kredi kartı hilelerini saptama, zeki araçlar ve robotlar için optimum rota belirleme, mekanik parçaların ömürlerinin tahmin edilmesi, ses tanıma, denetim, meteorolojik yorumlama, elektrik işareti tanıma, el yazısı tanıma, hastalıkların tanımlanması ve tedavisi, radar ve sonar sinyalleri sınıflandırma, spam maillerin filtrelenmesi olarak örnekler verilebilir.[16]

2.DERİN ÖĞRENME TABANLI NESNE TANIMA VE SEZME İLE İLGİLİ YAPILAN ÇALIŞMALAR VE ARAŞTIRMALAR

2.1 Derin Öğrenme

2.1.1 Yapay Sinir Ağı Öğrenme Özellikleri

İyi bir YSA minimum hataya sahiptir, öğrenme fonksiyonu YSA'yı yüksek hatalı bir konfigürasyondan düşük hatalıya taşır. YSA 'deki hata, hata ölçüm fonksiyonları ile ölçülür. Bu hata fonksiyonları eğitim kümesindeki örneklerin hatalarının tümünü temsil eder. Problemin tipine ve tasarımcının seçimine göre farklı hata fonksiyonları kullanılabilir.



Şekil 1. Yapay Sinir Ağı Gösterim

Katmanlar, ANN mimarisi katmanlardan kurulmuştur. Katmanların sayısı ANN'nin işlemsel karmaşıklığını ifade etmektedir. Daha çok sayıda katmandan oluşan ANN'ler her zaman daha az katmandan oluşanlara göre daha fazla işlem zamanı talep edeceklerdir.

Giriş katmanı (1), Gizli katman (n), Çıkış katmanı (1).

Giriş Katmanı, YSA'ya dış dünyadan bilgilerin geldiği katmandır. Bu katmandaki nöronların sayısı giriş sayısı kadardır. Her nöron bir girişe aittir. girdiler herhangi bir işleme uğramadan gizli katmana iletilirler.

Gizli katmanlar**,** Giriş katmanından aldığı bilgiyi işleyerek bir sonraki katmana iletir Bu katman asıl bilgi işlemenin gerçekleştirildiği katmandır. Her gizli katmandaki

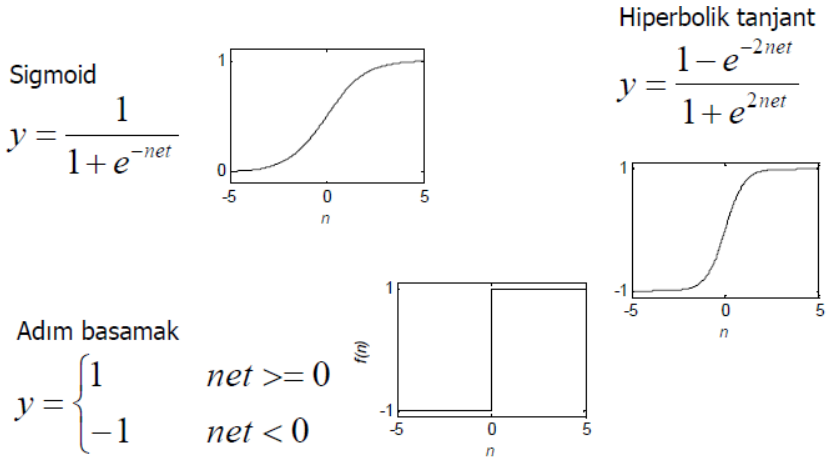
nöron sayısı değişkenlik gösterebilir. Genelde bu katmandaki nöronların sayısı giriş ve çıkış katmanındaki nöron sayılarından fazladır. Gizli katman sayısı ağın karmaşıklığını da kontrol altında tutmaya yarar.

Çıkış katmanı, Gizli katmandan gelen bilgiyi işler ve giriş katmanına gelen girdiye uygun olarak üretilen çıktıyı dış dünyaya gönderir.Nöronların sayısı sistemden beklenen çıkışlar kadardır. Her nöron tek bir çıkışa aittir.

Ağırlıklar, YSA eğitimi ağırlıklardaki değişim demektir. Ağırlıkların farklı kombinasyonları YSA'nin farklı performanslarını temsil edecektir. Ağırlıklar YSA zekasını oluşturur. YSA'nin öğrenme yeteneğinden bahsediyorsak ağrılıklarından bahsediyoruzdur.

Aktivasyon fonksiyonu: bir değişkeni farklı bir boyuta taşıyan doğrusal veya doğrusal olmayan bir fonksiyondur. Aktivasyon fonksiyonunun türevinin kolay alınabilmesi eğitim hızını arttıran bir özelliktir.

Sık kullanılan üç aktivasyon fonksiyonu:



Şekil 2. Aktivasyon Fonksiyonları Gösterim

Yapay Sinir Ağı Çalışma Adımları

- Örneklerin belirlenmesi
- Ağın topolojisinin belirlenmesi
  - Girdi ve çıktı sayısının belirlenmesi
- Ağın öğrenme parametrelerinin belirlenmesi

    - Öğrenme katsayısı ve sabitlerin belirlenmesi
- Ağın başlangıç değerlerinin atanması
- Epoch sayısı kadar
- Eğitim setindeki tüm örnekler için
    - Örnek ağa gösterilir
    - Hatanın hesaplanması
    - Bulunan hataya göre ağırlıkların güncellenmesi

2.2 Evrişimsel Sinir Ağı

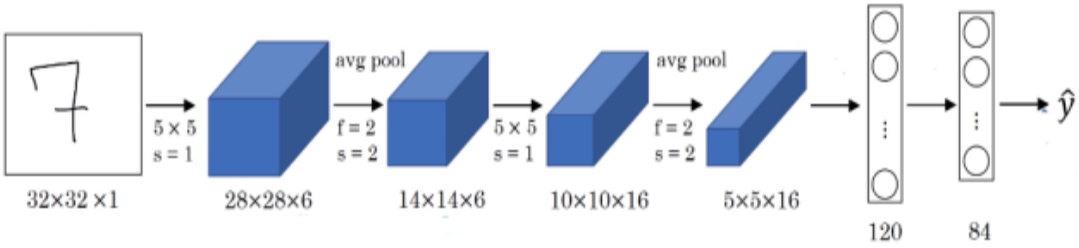
Verinin büyümesi ve veriden daha anlamlı bilgilere ulaşmak, öznitelik kestirimleriyle ilgili optimizasyon yapmayı zorunlu kılmaktadır. Klasik bir yapay sinir ağı modelleriyle nöronlar ve katmanlar arasındaki bağlantılar ve öğrenilen parametreler çok büyük hesaplama zorlukları ortaya çıkarmaktadır. İşte tam bu noktada Evrişimsel Sinir Ağlarına ihtiyacımız oluyor.

Konvolüsyonel Nöral Ağlar (CoNN veya ConvNet olarak, Türkçede ise evrişimsel sinir ağlar olarak da bilinirler), geleneksel nöral ağların modifiye edilmesi ile ortaya çıkmış bir kavramdır. Bu tür ağlar geniş ve derin yapıda olması sebebiyle derin nöral ağların veya derin öğrenmenin bir türüdür. Konvolüsyonel ağlar, resim tabanlı nesnelerin sınıflandırılmasında elde ettikleri büyük başarılar sebebiyle şu günlerde popülerdir.

Geleneksel nöral ağlar ile bir resmi sınıflandırmak ya da tanımak istiyorsanız tüm piksellerin nöral ağa aktarılması gerekmektedir. Konvolüsyonel ağlar da ise ilk önce resim üzerinde bazı örüntüler tespit edilmeye çalışılır, ve bu örüntüler nöral ağa aktarılırlar. Bu şekilde daha az kompleks şekilde resmi işleyebiliyorken daha başarılı sonuçlar elde edebiliyoruz.[12]

2.2.1 Evrişimsel Sinir Ağları Tarihçesi ve Methodları

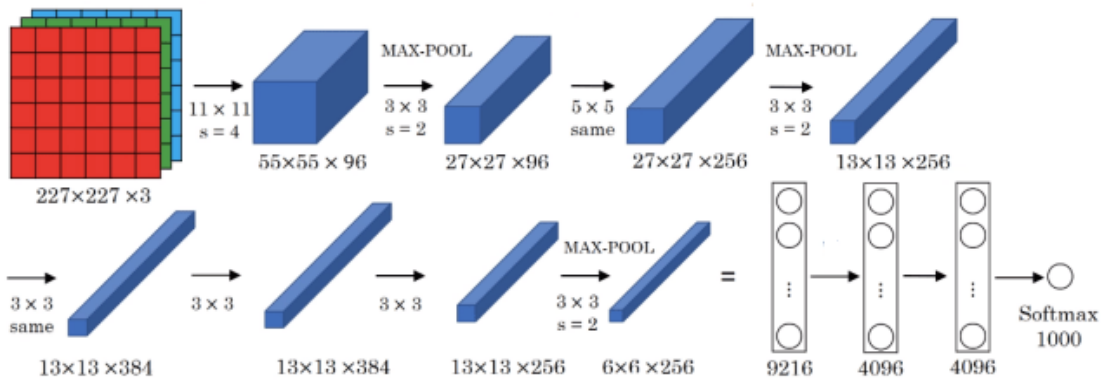
LeNet-5, 1998 yılında yayınlanmış ve ilk başarılı sonucu veren evrişimsel sinir ağı modelidir. Yann LeCun ve ekibi tarafından posta numaraları, banka çekleri üzerindeki sayıların okunması için geliştirilmiştir. MNIST (Modified National Institute of Standards and Technology) veri seti üzerinde deneyler gösterilmiştir. Bu modelde sonradan geliştirilecek diğer modellerden farklı olarak boyut azalma adımlarında max-pooling yerine average (ortalama) pooling işlemi yapılmaktadır. Ayrıca aktivasyon fonksiyonu olarak ta sigmoid ve hiperbolik tanjant kullanılmaktadır. [18]



Şekil 3. LeNet Gösterim

FC (Tam Bağlantı-Fully Connected) katmanına giren parametre sayısı 5x5x16=400 ve y çıkışında 0-9 arasındaki rakamları sınıflandırdığı için 10 sınıflı softmax bulunmaktadır. Bu ağ modelinde 60 bin parametre hesaplanmaktadır. Ağ boyunca matrisin yükseklik ve genişlik bilgisi azalırken derinlik (kanal sayısı) değeri artmaktadır.

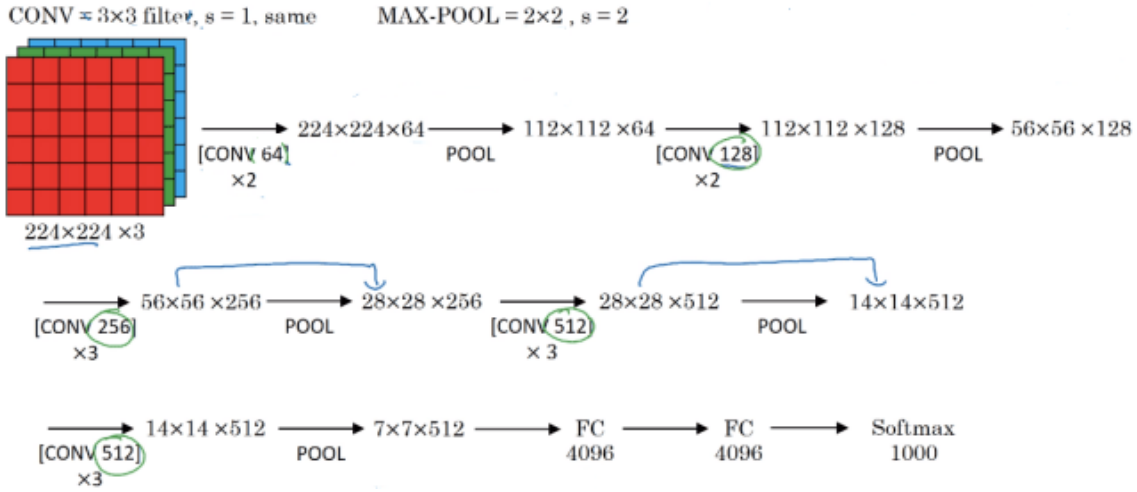
AlexNet, 2012 yılında evrişimsel sinir ağ modellerini ve derin öğrenmenin tekrar popüler hale gelmesini sağlayan ilk çalışmadır. Alex Krizhevsky, Ilya Sutskever and Geoffrey Hinton tarafından geliştirilmiştir. Temel olarak LeNet modeline, birbirini takip eden evrişim ve pooling katmanları bulunmasından dolayı, çok benzemektedir. Aktivasyon fonksiyonu olarak ReLU (Rectified Linear Unit), pooling katmanlarında da max-pooling kullanılmaktadır. [1]



Şekil 4. AlexNet Gösterim

Daha büyük ve daha derin olan bu ağ modeli paralel çift GPU (Grafik İşlem Birimi-Graphics Processing Unit) üzerinde iki parçalı bir modeldir. Yaklaşık olarak 60 milyon parametre hesaplanmaktadır. ImageNet ILSVRC yarışmasında sınıflandırma doğruluk oranını %74,3'ten %83,6'ya ani bir yükselme sağlayarak görüntü sınıflandırma probleminde bir kırılma noktasıdır.

VGG-16, basit bir ağ modeli olup öncesindeki modellerden en önemli farkı evrişim katmalarının 2'li ya da 3'li kullanılmasıdır. Tam bağlantı (FC) katmanında 7x7x512=4096 nöronlu bir öznitelik vektörüne dönüştürülür. İki FC katmanı çıkışında 1000 sınıflı softmax başarımı hesaplanır. Yaklaşık 138 milyon parametre hesabı yapılmaktadır. Diğer modellerde olduğu gibi girişten çıkışa doğru matrislerin yükseklik ve genişlik boyutları azalırken derinlik değeri (kanal sayısı) artmaktadır.[2]



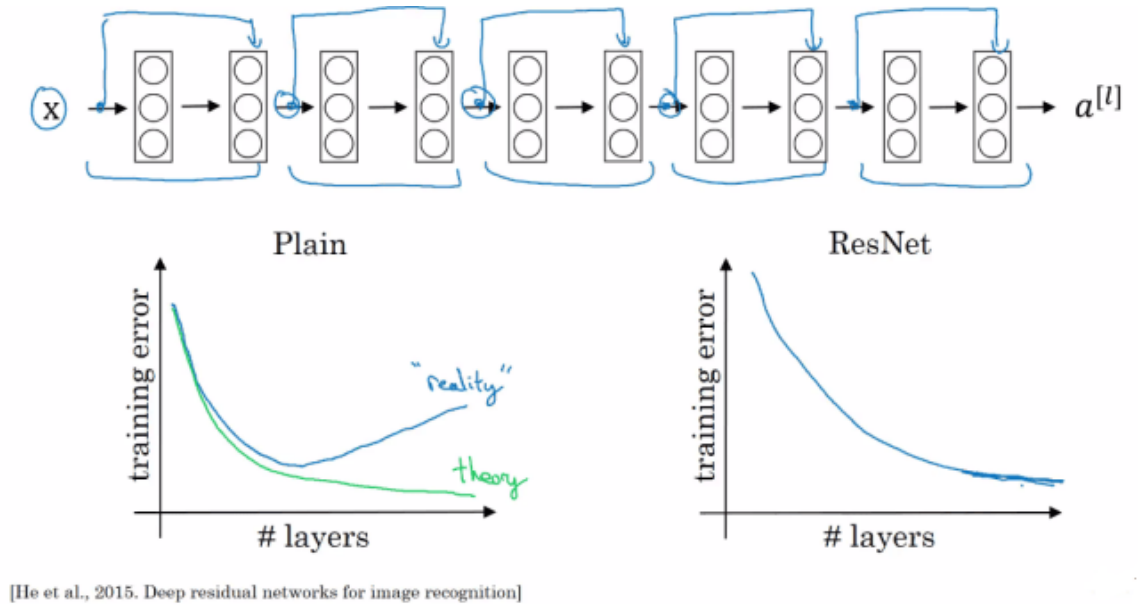
Şekil 5. VGG-16 Gösterim

Modelin her evrişim katmanı çıkışında farklı ağırlıklara sahip filtreler hesaplanır ve katman sayısı artıkça filtrelerde oluşan öznitelikler görüntünün 'derinliklerini' simgelemektedir.

ResNet, ağ modelinin gerçek anlamda derinleşmeye başladığı kendinden önceki modellerden farklı bir mantığı barındıran ResNet; artık değerlerin (residual value) sonraki katmanlara besleyen blokların (residual block) modele eklenmesiyle oluşmaktadır. ResNet bu özelliği ile klasik bir model olmaktan çıkmaktadır. [3]

Doğrusal ve ReLU arasında iki katmanda bir eklenen bu değer aşağıdaki gibi sistemdeki hesabı değiştirir. Önceden gelen a[l]değeri, a[l+2] hesabına eklenmiş olur.

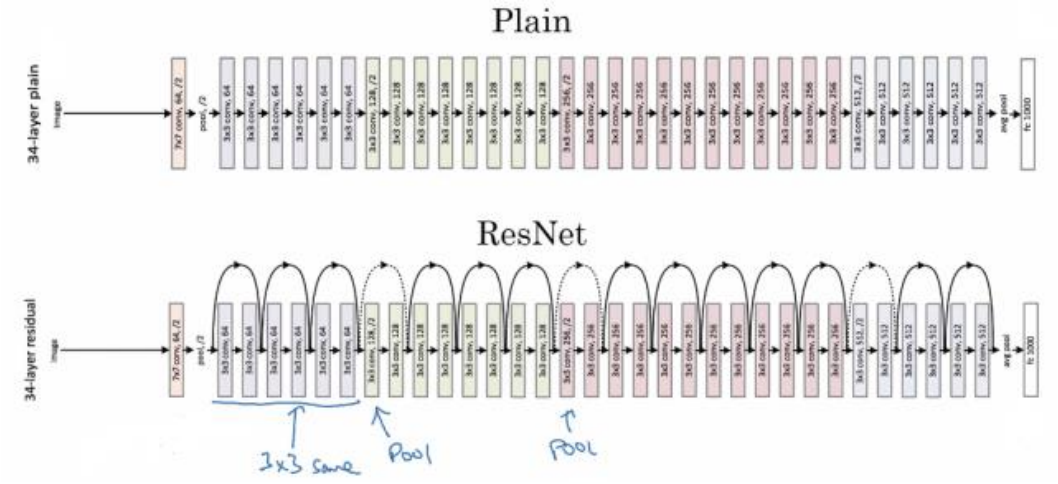
Teorik olarak, modelde katman sayısı arttıkça başarımın artacağı düşünülür. Ancak gerçekte böyle olmadığı deneyimlenmiştir. Buradan hareketle ResNet modeli oluşturulmuştur. Böylece $w^{[l+2]}=0$ olduğu durumda yeni teoriye göre $a^{[l+2]}=b^{[l+2]}$ olur. Bu (vanishing gradient) istenmeyen bir durumdur.



[He et al., 2015. Deep residual networks for image recognition]

Şekil 6. ResNet Gösterim

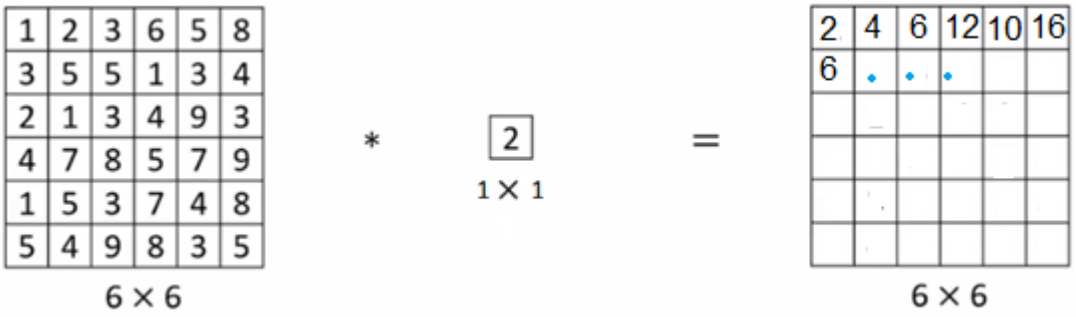
Ancak artık değer (residual) beslemesi yeni çıkış eşitliğini iki önceki katmandan gelen $a^{[l]}$değeri o an ki ağırlık 0 olsa bile öğrenme hatasını optimize eder. Daha hızlı eğitilir.



Şekil 7. ResNet Gösterim-1

Ağ içerisinde ağ, 2013 yılında Min Lin ve arkadaşlarının "Network in Network" makalesi ile modellerdeki hesaplama karmaşıklığına yeni ve ses getiren bir çözüm önermiştir. Google, bu öneri sayesinde sunulan modeli hemen kullanmıştır ve başarıya ulaşmıştır. Hesaplamada esneklik sağlanmış ve başarımı artırmak için varyasyonlu modellerin tasarlanmasına olanak tanınmıştır.[4]

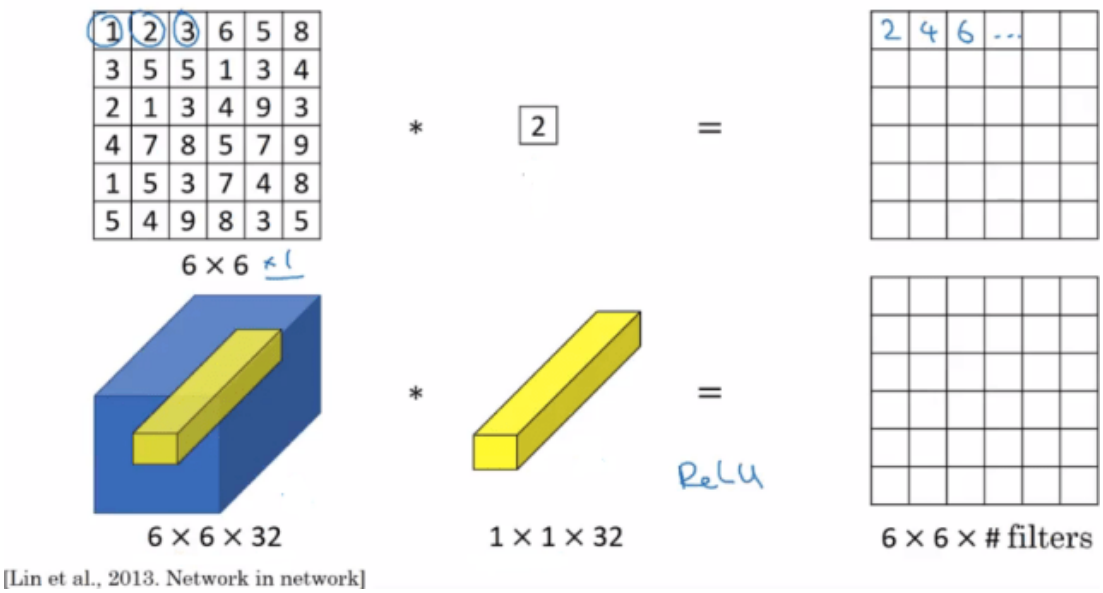
Peki ağ içinde ağ ne demek? Aslında son derece basit bir matematiktir: 1×1 evrişim işlemi Ancak 1×1 element işlemin matris üzerinde bir etkisi olmaz diye düşünülebilir.



Şekil 8. Ağ içinde Ağ Gösterim

Ancak giriş matrisi çok kanallıysa durum değişir. Örneğin giriş matrisi 100 kanallıysa ve buna 30 kanallı bir 1×1 evrişim filtresi hacimsel olarak uygulandığında çıkış matrisinin kanal sayısı filtre sayısına eşit, yani 30 olur. O zaman 1×1 evrişim katmanı

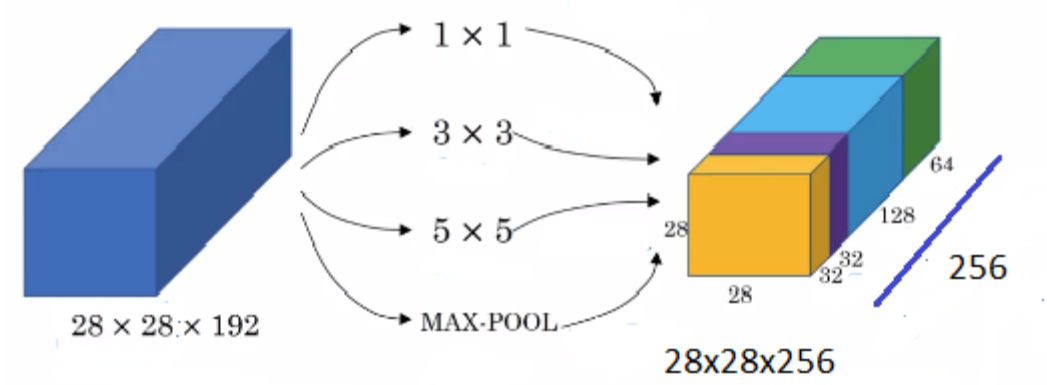
derinlikte boyut azaltmak demektir. Aşağıdaki örnekte de çıkış kanal sayısı filtre ile eşit şekilde hesaplanmaktadır.



Şekil 9. Ağ içinde Ağ Gösterim-1

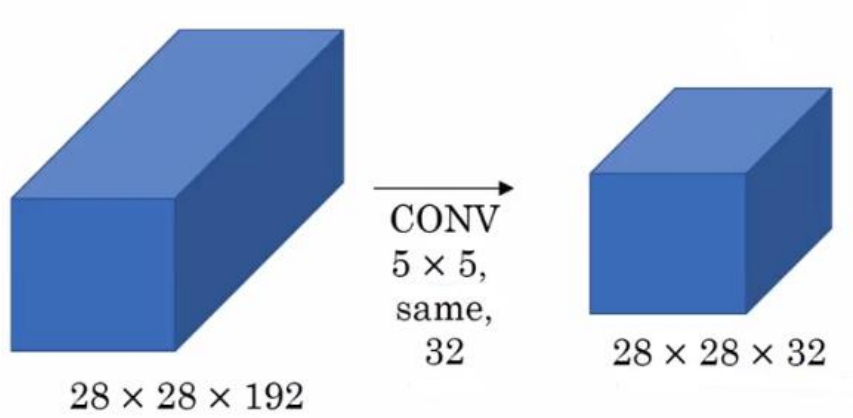
Klasik ağ modellerinin tamamında modelin başından sonuna doğru ilerlerken matrislerin yükseklik ve genişlikleri pooling katmanları sayesinde azaltılıyordu. Ancak kanal sayıları artıyordu. O halde neden 1×1 evrişim katmanlarını modellere ekleyip kanal sayısını da istediğimiz gibi sınırlandırmayalım?

Inception Ağlar,şu ana kadar bahsi geçen ağ modellerinden farklı ve anlaşılması zor olanıdır. Ancak hesaplama karmaşıklığına ve büyüklüğüne bulunan çözüm beraberinde hız ve başarımı getirmektedir. En kolayı bunu bir örnekle açıklamaktır.



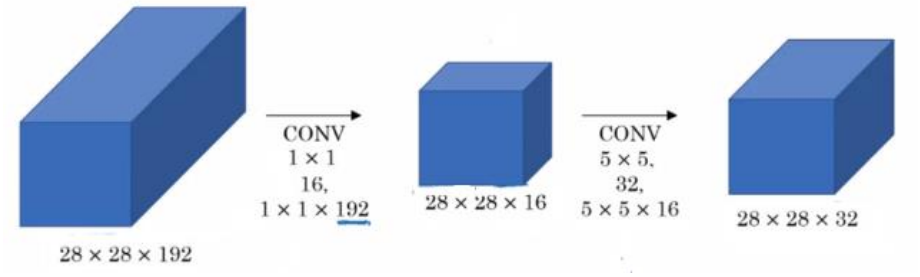
Şekil 10. GoogleNet Gösterim

Inception ağ modeli modüllerden oluşmaktadır. Her bir modül, farklı boyutlu evrişim ve max-pooling işlemlerinden meydana gelmektedir. Yukarıdaki görüntüde 3 farklı evrişim ve bir max-pooling işlemi sonucunca 28x28x256 boyutlu bir tensör elde edilmiştir. Bu çıkıştan sadece 5×5 evrişim işlemindeki parametre sayısını hesaplayıp işlem kompleksliğini değerlendirelim.



Şekil 11. GoogleNet Gösterim-1

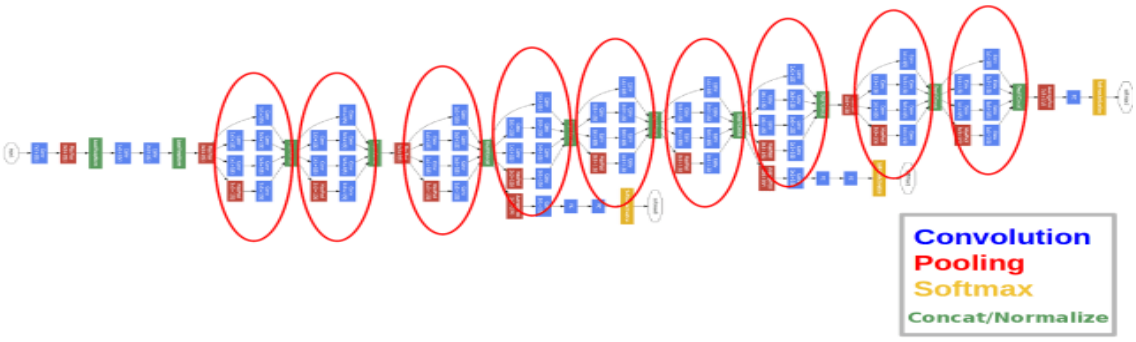
Yalnızca bu işlem adımı için (28x28x32)x(5x5x192)=120 milyon parametre hesaplanması gerekir. Bunun gibi diğer evrişim ve max-pooling katmanlarını da aynı şekilde hesaplamak gerekir. C. Szegedy ve ekibi 'Network in Network' makalesine atıf yaparak her evrişim katmanından önce 1×1 evrişim katmanı kullanılarak işlem yükünü optimize etmeye odaklanmıştır. [7]



Şekil 12.GoogleNet Gösterim-2

Böylece daha karmaşık bir ağ modelin ile daha az hesap ve daha hızlı bir tasarım yapılmaktadır. Bu koşulda; 1×1 evrişim katmanında: (28x28x16)x(1x1x192)=2,4 milyon parametre ve 5×5 evrişim katmanında: (28x28x32)x(5x5x16)=10 milyon parametre olmak üzere toplamda 12.4 milyon parametre hesaplanmaktadır. İlk duruma göre yaklaşık 10 kat daha az parametre hesabı son derece çarpıcıdır. Bu 1×1 evrişim işlemini 'bottleneck' (darboğaz) olarak tanımlamışlardır.

Her bir modüle ‘inception’ adı verilmektedir. Toplam 9 inception bloğundan oluşan modele, bu işlere (derin öğrenme) başlangıç değeri veren LeNet modeline atıf yaparak GoogLeNet adı verilmektedir. GoogLeNet modelinde modelin kendisi de genişlemektedir.



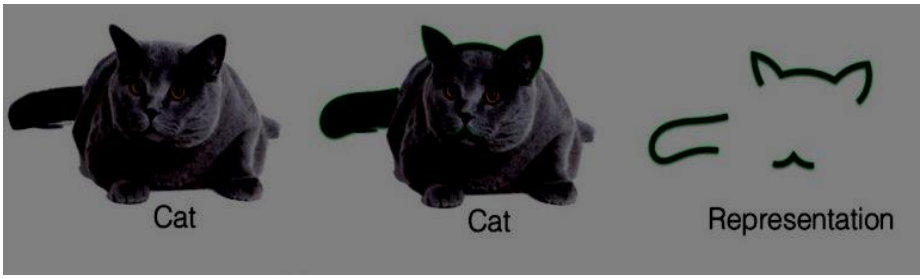
Şekil 13. GoogleNet Gösterim-3

2.3 Konvolüsyonel Nöral Ağı Ve Türleri

2.3.1 Konvolüsyonel Nöral Ağı(CoNN)

Geleneksel nöral ağlar ile bir resmi sınıflandırmak ya da tanımak istiyorsanız tüm piksellerin nöral ağa aktarılması gerekmektedir. Konvolüsyonel ağlar da ise ilk önce resim üzerinde bazı örüntüler tespit edilmeye çalışılır, ve bu örüntüler nöral ağa aktarılırlar. Bu şekilde daha az kompleks şekilde resmi işleyebiliyorken daha başarılı sonuçlar elde edebiliyoruz.

Elinizdeki resimleri kedi veya kedi değil şeklinde sınıflandırmak istediğinizi varsayalım. En genel haliyle, bilinen kedi resimlerinden kedilerin kulak, ağız ve kuyruk şekilleri çıkartılır. Daha sonra bu şekiller yeni resimlerde var mı yok mu şeklinde aranırlar. Konvolüsyonel ağlarda bu şekillere filtre denmektedir.[13][14]



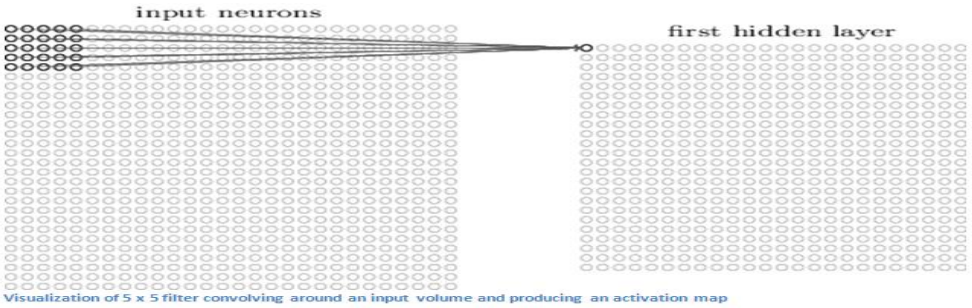
Şekil 14. CoNN Gösterim

Böylelikle bu örüntüleri içeren resimleri kedi, içermeyenleri ise kedi değil şeklinde sınıflandıracağız.

Şekil 15. CoNN Gösterim-1

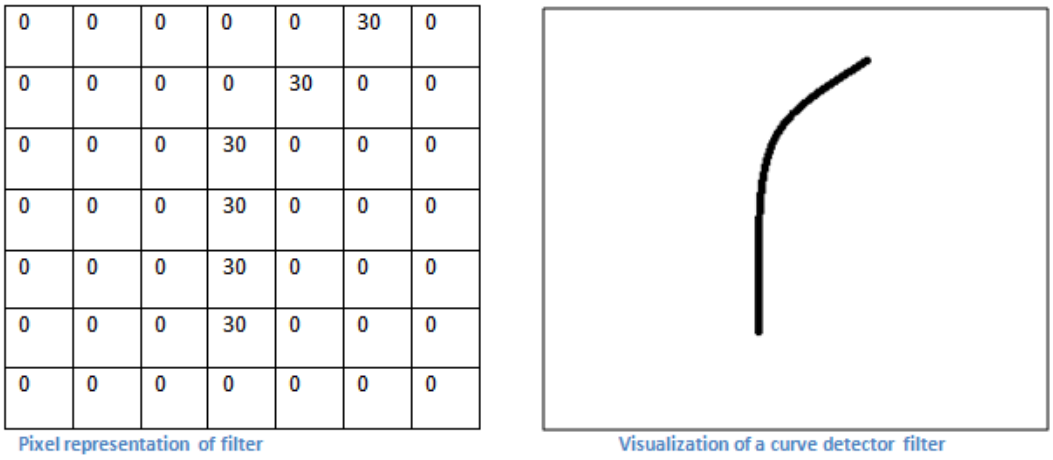
Konvolüsyon Katmanı,bu katmanda resim filtre boyutuna göre daha küçük bir boyuta indirgenir. Örneğin 32x32x3 pixel boyutundaki bir resme 5x5x3 boyutunda bir filtre uygulanması ile 3×3 boyutunda bir resim elde edilir. Bu filtrenin derinliği ile girişteki boyutun derinliği aynı olmalıdır.Filtre sol üst köşeden başlayıp, kayarak filtredeki değerleri resimdeki piksel değerleri ile çarpılır.Bu çarpımların hepsi toplanır tek bir piksele(sol üstteki piksele) denktir sonra her bir konum için tekrarlanır.28x28x3 değerinde bir konvolüsyondan geçmiş matris elde edilir.

Konvolüsyon Katmanı sonrası çıktı hesabı:
(((input boyutu-filtre boyutu)+2*padding)/stride(atlama boyutu)) +1' dir



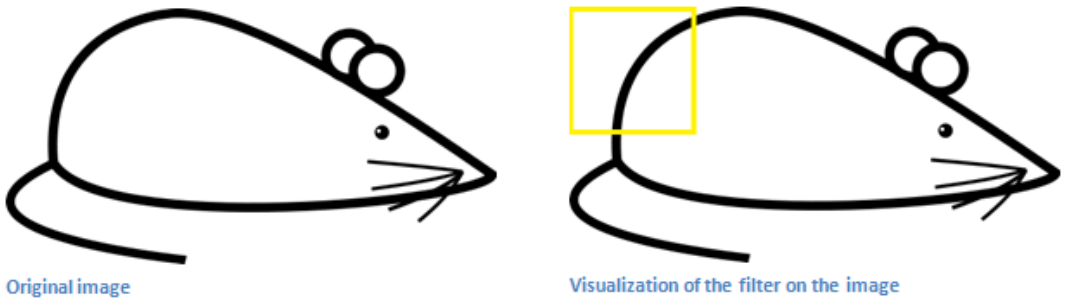
Şekil 16. Konvolüsyon Katman

Bu filtrelerin her biri, özellik tanımlayıcıları olarak düşünülebilir. Özellikler dediğimde, düz kenarlar, basit renkler ve eğriler gibi şeyler. Tüm görüntülerin birbiriyle ortak olduğu en basit özellikler. İlk filtremizin 7 x 7 x 3 olduğunu ve bir eğri dedektörü olacağını varsayalım. Bir eğri detektörü olarak, filtre içinde bir piksel yapısına sahip olacaktır. Bir eğri şekli olan alan boyunca daha yüksek sayısal değerler (Unutmayın, bu filtrelerden sadece rakamlar olarak bahsediyoruz!).



Şekil 17. Konvolüsyon Katman-1

Bu filtreyi, giriş hacminin sol üst köşesinde bulundurduğumuzda, bu bölgedeki filtre ve piksel değerleri arasında çarpımlar hesaplanır. Şimdi, sınıflandırmak istediğimiz bir resmin örneğini alalım ve filtremizi sol üst köşeye koyalım.[12]



Şekil 18. Konvolüsyon Katman-2

Yapılması gereken, filtredeki değerleri görüntünün orijinal piksel değerleri ile çarpmaktır.

Visualization of the receptive field

0	0	0	0	0	0	30
0	0	0	0	50	50	50
0	0	0	20	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0
0	0	0	50	50	0	0

Pixel representation of the receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = (50*30)+(50*30)+(50*30)+(20*30)+(50*30) = 6600 (A large number!)

Şekil 19. Konvolüsyon Katman-3

Temel olarak, giriş görüntüsünde, genellikle bu filtrenin temsil ettiği eğriye benzeyen bir şekil varsa, birlikte toplanan çarpımların hepsi büyük bir değere neden olur.Fakat filtre hareket edildikçe gelen sonuçlar değişecektir.Çünkü aşağıdaki gibi filtre kendisine karşılık gelmeyen resmin farklı kısımlarıyla eşleşecektir ve değerler düşük çıkacaktır.

Visualization of the filter on the image

0	0	0	0	0	0	0
0	40	0	0	0	0	0
40	0	40	0	0	0	0
40	20	0	0	0	0	0
0	50	0	0	0	0	0
0	0	50	0	0	0	0
25	25	0	50	0	0	0

Pixel representation of receptive field

*

0	0	0	0	0	30	0
0	0	0	0	30	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	30	0	0	0
0	0	0	0	0	0	0

Pixel representation of filter

Multiplication and Summation = 0

Şekil 20. Konvolüsyon Katman-4

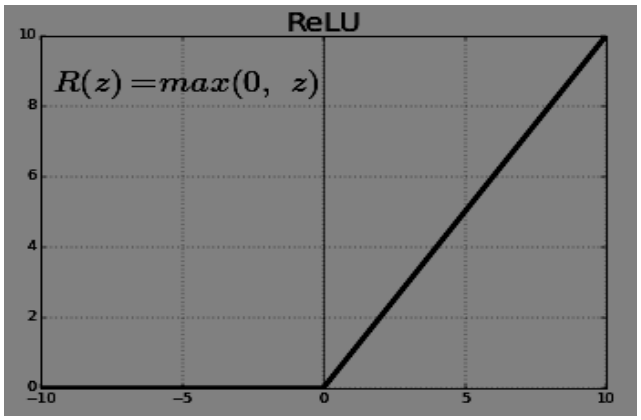
Bu konvolüsyon tabakasının çıktısı bir aktivasyon haritasıdır. Dolayısıyla, bir filtre konvolüsyonunun basit durumunda (ve eğer bu filtre bir eğri dedektörü ise), aktivasyon haritası, çoğunlukla resimde eğrilerin bulunacağı alanları gösterecektir. Bu örnekte, 28 x 28 x 1 etkinleştirme haritanızın sol üst değeri 6600 olacaktır. Bu yüksek değer, giriş hacminde filtrenin etkinleşmesine neden olan bir çeşit eğri olabileceği anlamına gelir. Etkinleştirme haritanızdaki en üstteki değer 0 olur, çünkü giriş hacminde filtrenin etkinleştirilmesine neden olan hiçbir şey yoktu (veya daha basit bir ifadeyle, orijinal görüntünün o bölgesinde bir eğri yoktur). Bu sadece bir filtredir. Bu sadece dışarı doğru ve sağa eğri çizgiler algılayacak bir filtredir. Sola veya düz

kenarlara eğri çizgiler için başka filtrelerimiz de olabilir. Daha fazla filtre, aktivasyon haritasının derinliği ve giriş hacmi hakkında sahip olduğumuz daha fazla bilgi. İlk katmanda, girdi sadece orijinal görüntüydü. Bununla birlikte, 2. konvolüsyon katmanında, giriş, ilk katmandan kaynaklanan aktivasyon haritasıdır. Dolayısıyla, girişin her bir katmanı, temel görüntüdeki belirli düşük düzey özelliklerin göründüğü yerleri temel olarak açıklar. Artık bunun üstüne bir filtre uyguladıktan sonra, çıktı daha yüksek düzey özellikleri temsil eden aktivasyonlar olacaktır. Bu özellikler bir eğri ve düz kenar kombinasyonu veya kareler (birkaç düz kenarın kombinasyonu) olabilir. Ağ üzerinden ilerledikçe ve daha fazla dönüşüm katmanı yaptığınızda, daha fazla karmaşık özelliği temsil eden aktivasyon haritaları elde edersiniz. Ağın sonunda, görüntüde el yazısı olduğunda etkinleştiren bazı filtreler, pembe nesneleri gördüklerinde etkinleşen filtreler vb. olabilir.[10]

Input -> Conv -> ReLU -> Conv -> ReLU -> Pool -> ReLU -> Conv -> ReLU -> Pool ->Fully Connected

Aktivasyon Katmanı, ReLU fonksiyonu diğer yaygın aktivasyon fonksiyonlarından lineer olması sebebiyle daha hızlı çıktılar üretebilmektedir. Performans önemli bir kıstas olduğu için konvolüsyonel ağlarda ReLU sıklıkla kullanılmaktadır.[12]

Eğer nöron ağımızı binary sınıflandırma(öyle veya değil) şekilinde değil çoklu sınıflandırma şeklinde kullanmak istiyorsak Softmax Aktivasyon Fonksiyonu kullanılır.

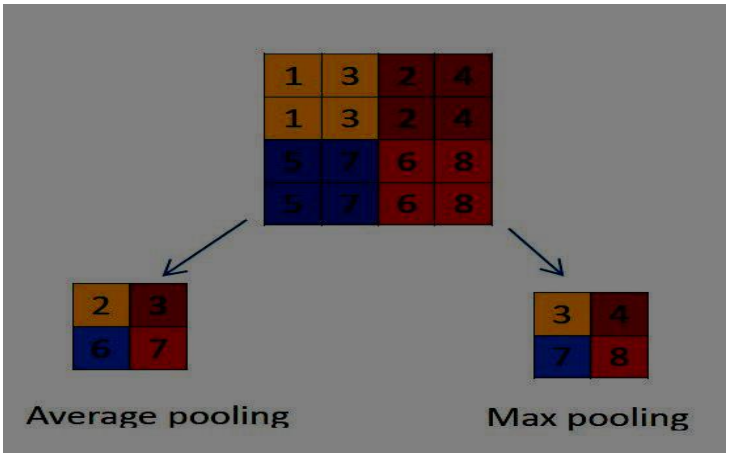


Şekil 21. Relu

Matematiksel olarak aşağıdaki şekilde gösterilebilir.Konvolüsyon yapıldıktan sonra değerlerin sıfırın altında olduğu görülmektedir.Bunları Relu ile değerlerini değiştirerek düzenleriz.

Şekil 22. Relu-1

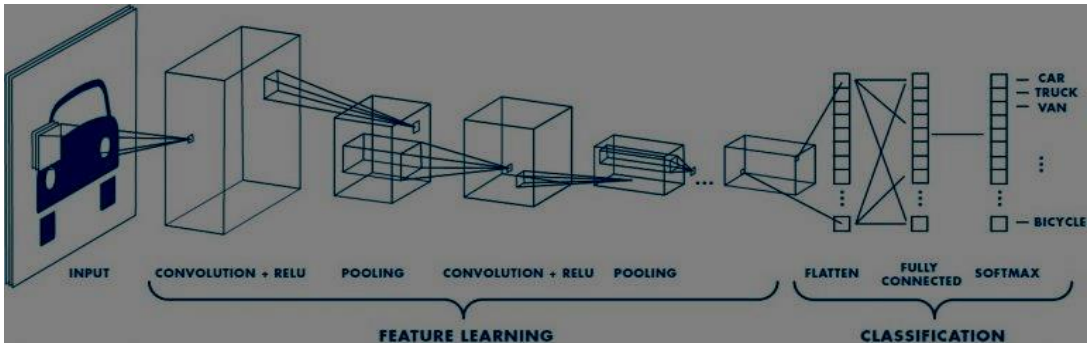
Maksimum Havuzlama katmanında, elde edilen matrisler genişlik ve uzunluk olarak yeniden daha düşük boyutlara indirgenirler ama derinlik sabit kalır. Ağın bu katmanında öğrenilen parametre yoktur. Hesaplama karmaşıklığını azaltmak için kullanılan bir adımdır.İstenilen Filtre boyutundaki maksimum, eleman yeni matriste saklanır. Bununla birlikte maksimum yerine çerçevenin ortalama veya minimum değerinin saklanması ile ortalama havuzlama işlemi de yapılabilir. Ancak Hinton'ın kapsül teorisine göre verideki önemli bazı bilgilerinde kaybolmasına sebep olduğu için başarımdan ödün vermektedir. Özellikle lokasyon bilgisinin çok önemli olmadığı problemlerde yine de oldukça güzel sonuçlar vermektedir.



Şekil 23. Havuzlama Katmanı Gösterim

Konvolüsyon, aktivasyon ve havuzlama adımları tekrar tekrar uygulanarak konvolüsyon nöral ağlar oluşturulabilir. Bu işlemler neticesinde indirgenmiş matristeki değerler geleneksel nöral ağa girdi olarak verilirler.

Tam Bağlantılı Katman(Fully Connected Layer),birden çoklu konvolüsyon ve havuzlama katmanından geçtikten sonra en sonunda bu layerda görüntünün ne olduğuna karar verilir.Temel olarak, bir FC katmanı, belirli bir sınıfla en güçlü şekilde ilişkili olan yüksek seviyeli özelliklere bakar ve Ağırlıklar ile önceki katman arasındaki ürünleri hesaplarsanız, farklı sınıflar için doğru olasılıkları elde edersiniz.Backpropagation burada gerçekleşir. Eğer program bir resmin bir köpek olduğunu tahmin ediyorsa, bir pençe veya 4 bacak gibi yüksek seviyeli özellikleri temsil eden aktivasyon haritalarında yüksek değerlere sahip olacaktır. Benzer şekilde, eğer program bazı görüntülerin bir kuş, kanatlar veya gaga vb. gibi üst düzey özellikleri temsil eden aktivasyon haritalarında yüksek değerlere sahip olacaktır. Temel olarak, bir FC katmanı, belirli bir sınıfla en güçlü şekilde ilişkili olan yüksek seviyeli özelliklere bakar ve ağırlıklar ile önceki katman arasındaki ürünleri hesaplarsanız, farklı sınıflar için doğru olasılıkları elde edersiniz.



Şekil 24. Tam Bağlantılı Katman Gösterim

Bu tür ağlar sadece örüntülerin var olup olmadığını kontrol eder. Örüntülerin anlamlı olup olmaması konvolüsyonel ağlar için bir şey ifade etmemektedir.Özetle konvolüsyonel nöral ağlar, resim tabanlı nesnelerin analizinde sıklıkla kullanılan kuvvetli ama mükemmel olmayan bir derin öğrenme yöntemidir.

Eğitim, CoNN başlamadan önce, ağırlıklar veya filtre değerleri randomize edilir. Filtreler kenarları ve eğrileri aramayı bilmez.Yani başta filtreler pençeleri ve gagaları aramayı bilmezler.Bir resim ve bir etiket verilmesi fikri, CoNN'lerin geçtiği eğitim sürecidir. Binlerce köpek, kedi ve kuş resmine sahip bir eğitim setimiz olduğunu ve resimlerin her birinin, resmin hangi hayvan olduğuna dair bir etiketi olması gerekir.

Geriye doğru yayılma(Backpropagation), 4 ayrı bölüme, ileri geçişe, kayıp fonksiyonuna, geri geçişe ve ağırlık güncellemesine ayrılabilir. İleri geçiş sırasında, hatırladığımız gibi 32 x 32 x 3 sayı dizisi olan ve tüm ağ üzerinden ileten bir eğitim görüntüsü alırsınız. İlk eğitim örneğimizde, tüm ağırlıklar veya filtre değerleri rastgele başlatıldığı için, çıktı büyük olasılıkla [.1 .1 .1 .1 .1 .1 .1 .1] gibi bir şey olacaktır. Özellikle herhangi bir sayıya tercih vermeyen çıktı.Ağ, mevcut ağırlıkları ile, bu düşük seviyeli özelliklere bakamaz veya bu nedenle sınıflandırmanın ne olabileceği konusunda makul bir sonuca ulaşamaz. Bu, geri yayılımın kayıp fonksiyonu bölümüne gider. Şu an kullandığımız şey eğitim verileridir. Bu veriler hem bir görüntü hem de bir etikete sahiptir. Örneğin, girilen ilk eğitim görüntüsünün 3 olduğunu varsayalım. Görüntünün etiketi [0 0 0 1 0 0 0 0 0 0] olacaktır. Bir kayıp fonksiyonu birçok farklı şekilde tanımlanabilir, ancak ortak olanı MSE'dir .

Ortalama karesel hata= ½ (gerçek – tahmin edilen) karedir.

$$E_{total} = \sum \frac{1}{2}(target - output)^2$$

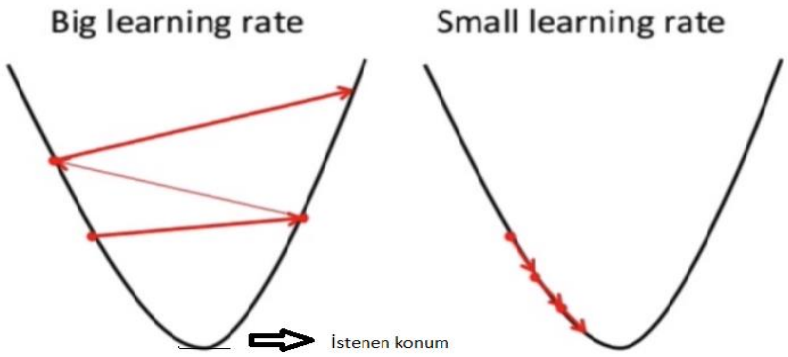
İlk iki eğitim görüntüsü için kayıp çok yüksek olacaktır. Şimdi, bunu sezgisel olarak düşünelim. Tahmin edilen etiketin eğitim etiketi ile aynı olduğu bir noktaya varmak istiyoruz.

Bu, W'nin belirli bir katmandaki ağırlıklar olduğu bir dL / dW'nin matematiksel karşılığıdır. Şimdi, yapmak istediğimiz, ağda geriye doğru bir geçiş yapmaktır; bu, hangi ağırlıkların kayıplara en çok katkıda bulunduğunu ve bu kayıpların azalması için bunları ayarlama yollarının bulunduğunu belirlemektedir. Bu türevi hesapladıktan sonra, ağırlık güncellemesi olan son adıma geçiyoruz. Bu, filtrelerin tüm ağırlığını aldığımız ve bunları degrade yönünde değişecek şekilde güncellememizdir.

$$w = w_i - \eta \frac{dL}{dW}$$

w = Weight
w_i = Initial Weight
η = Learning Rate

Öğrenme hızı programlayıcı tarafından seçilen bir parametredir. Yüksek öğrenme oranı, ağırlık güncellemelerinde daha büyük adımların atıldığı anlamına gelir ve bu nedenle, modelin optimal bir ağırlık setine yaklaşması daha az zaman alabilir. Bununla birlikte, çok yüksek bir öğrenme oranı, çok büyük ve optimum noktaya ulaşmak için yeterince hassas olmayan sıçramalara neden olabilir.Birinci grafikte büyük sıçramalar görülür.İkincisinde ise istenilen şekilde öğrenme gerçekleşmektedir.



Şekil 25. Öğrenme Oranı Gösterim

İleri geçiş, kayıp işlevi, geriye doğru geçiş ve parametre güncellemesi genellikle tek yönlü olarak adlandırılır. Program, her eğitim görüntüsü için bu süreci sabit sayıda tekrarlayacaktır. Son eğitim örneğindeki parametre güncellemesini tamamladıktan sonra, ağın katmanların ağırlıklarının doğru ayarlanması için ağın yeterince iyi eğitilmesi gerektiğini umarız.

Test, son olarak, CoNN'mizin işe yarayıp yaramadığını görmek için farklı bir dizi resim ve etikete sahibiz(eğitim ve test arasında iki katına çıkamıyoruz!). Ve görüntüleri CoNN'den geçiriyoruz. Çıktıları temel gerçekle karşılaştırırız ve ağımızın işe yarayıp yaramadığını görebiliriz!

Gradient Descent, ağırlıklar, eğitilerek istenen en optimum konuma getirilmeye çalışılması.Ulaştığında öğrenmesini tamamlayıp doğru sonuçlar vericektir.Gradyan buradaki iniş demek.Hatanın düşüşünü gösteriyor.Stochastic Gradient Descendent adım adım veride ilerleyerek optimumu bulmaktır.Batch Gradient Descendent ise tüm verinin üzerinden optimumu bulmak.

Sigmoid, sınflandırma sonucu binary output ise loss hesaplamak için Sigmoid fonksiyonu kullanılmaktadır. Çok sınıflı problemler için Sigmoid uygun değildir.

Loss fonksiyonu, tasarlanan modelin hata oranını aynı zamanda başarımını ölçen fonksiyondur. Derin ağların son katmanı loss fonksiyonun tanımlandığı katmandır. Loss fonksiyonu, hata hesaplama işini problemi bir optimizasyon problemine dönüştürerek yaptığı için optimizasyon terminolojinde kullanılan objective function,

cost function isimleriyle de tanımlanmaktadır. Loss fonksiyonu temelde modelin yaptığı tahminin, gerçek değerden (ground truth) ne kadar farklı olduğunu hesaplamaktadır. Bu nedenle iyi tahmin eden bir model oluşturmamışsak, gerçek değer (ground turth) ile tahmin edilen değer arasındaki fark yüksek olacak dolayısıyla loss değeri yüksek olacak, iyi modele sahipsek loss değeri az olacaktır. Birebir aynı olduğu durumda loss 0 olacaktır. İyi bir modelden beklentimiz 0'a yakın loss değerine sahip olmasıdır.

Softmax fonksiyonu, yapay sinir ağı tarafından üretilen skor değerlerini kullanarak olasılık temelli loss değeri üretmektedir. Softmax sonucunda test girdisinin her bir sınıfa ait benzerliği için olasılık değeri üretilir. Yapay sinir ağı modelinin çıktı olarak verdiği skor değerler normalize edilmemiş değerlerdir. Softmax bu değerleri normalize ederek olasılık değerlerine dönüştürmektedir. Olasılık değerine dönüştürme işlemini istatistikte sıkça kullanılan en çok olabilirlik (maximum likelihood) fonksiyonuyla yapmaktadır. Değerler çok büyük değerler olduğu için likelihood fonksiyonun log'nu alarak dağılımın yapısını korumaktadır. Özetle Softmax fonksiyonunda amaç test verisinin doğru sınıfı için log likelihood değerini maksimize etmektir. Loss fonksiyonunda amaç ise negative likelihood minimize etmektir; yani en az farka sahip, en çok benzerliği aship sınıfı bulmaktır.

Epoch, tüm veri setinin sinir ağları boyunca bir kere gidip gelmesine(ağırlıkların güncellenmesi) epoch denir.Epoch'un arttırılması daha iyi verim sağlamayabilir tam tersine eğitim verisini ezberlemeye başlamasına neden olabilir ve yeni bir veride(eğitim verisinde olmayan) sonuçları istenilen performansta olmayabilir. Epoch'un az olması da istenilen performansa ulaştırmayabilir.

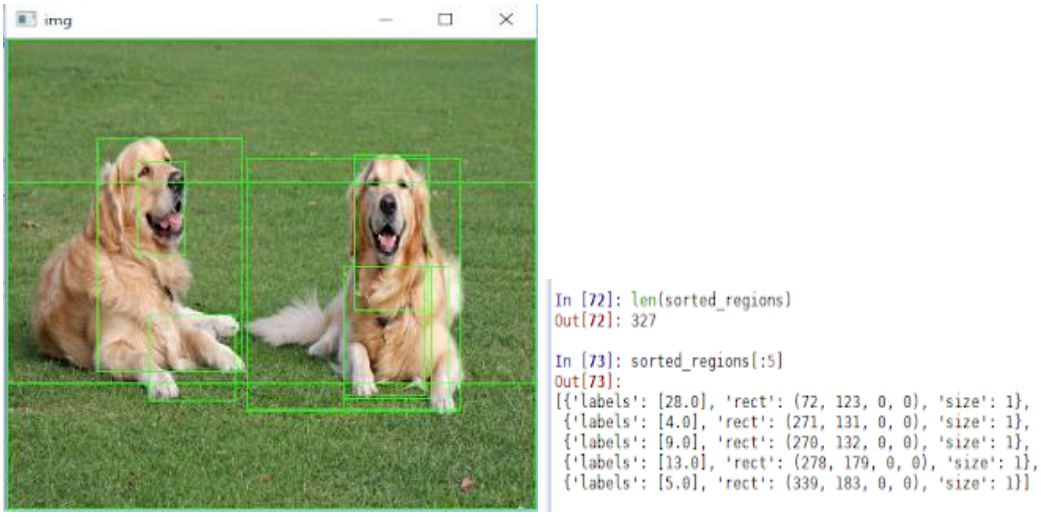
Batch, bir epoch içerisinde, veri seti içerisindeki bir veri dizisi sinir ağlarında sona kadar gider daha sonra orada bekler batch boyutu kadar veri sona ulaştıktan sonra hata oranı hesaplanır ve optimizasyon işlemi yapılır(ağırlıklar güncellenir).

2.3.2 R-CoNN(Region-Based Convolutional Neural Network)

Çok sayıda bölge seçme problemini atlamak için, görüntüden yalnızca 2000 bölgeyi çıkarmak için seçici bir arama kullandığımız ve bölge önerileri olarak adlandırdığımız bir yöntem önerildi. Bu nedenle, şimdi, çok sayıda bölgeyi sınıflandırmaya çalışmak yerine, yalnızca 2000 bölgeyle çalışabilirsiniz. Bu 2000 bölge önerileri, aşağıda yazılan seçici arama algoritması kullanılarak üretilmiştir.Bu 2000 adet bölge sonra ki aşamada boyutları eşit olucak şekilde düzenleniyor.Çünkü CoNN giriş olucak bu inputların boyutları eşit olmalı.[8]

Bölge Önerisi(Region Proposa), bölge önerme algoritması kabaca, girdi olarak aldığı bir görüntüdeki nesne bulunma ihtimali olan alanları (objectness) çevreleyen

kutucukları (alt görüntü alanlarını) hesaplar. Daha sonra ise bulunan bu alt alanlarda nesne tanıma modelleri kullanılarak sınıflama işlemi gerçekleştirilebilir.Image Segmentasyon hesaplama temelli bir yaklaşımdır. F&H algoritması ile başlangıç segmentlerini bulduktan sonra, her bir segment bölgesini bölge adayı olarak listeye ekler. Daha sonra ise benzer olan segmentleri gruplayarak segment adayı sayısını azaltır ve bu işlemi iteratif olarak tekrarlar. Her bir iterasyonda daha büyük bir bölge adayı bulunur ve aday listesine eklenir, böylece hiyerarşik bir yapıda küçükten büyüğe doğru bir segment listesi hazırlanmış olur.Bölgelerin olasılıklarına göre "object like" veya "not object-like" olarak çıkış alınır.



Şekil 26. Bölge Önerisi Gösterim

Seçici Arama(Selective Search), ilk alt bölümleme oluşturuluyor, birçok aday bölge üretiyoruz.Benzer bölgeleri tekrarlayan şekilde daha büyük alanlara birleştirmek için açgözlü(greedy) algoritmayı kullanıyoruz.Son aday bölge tekliflerini üretmek için oluşturulan bölgeleri kullanılıyor.[5]

Bölgeleri birlikte gruplamak için tek bir optimal strateji yoktur. Bölgeler sadece renk, doku nedeniyle veya parçalar kapalı olduğundan bir nesne oluşturabilir. Ayrıca, gölgeleme ve ışığın rengi gibi aydınlatma koşulları, bölgelerin bir nesneyi nasıl oluşturduğunu etkileyebilir. Bu nedenle çoğu durumda iyi sonuç veren tek bir strateji yerine, tüm vakalarla başa çıkmak için çeşitli stratejiler kullanılıyor.

İki en benzer bölge birlikte gruplanır ve yeni benzerlikler arasında hesaplanır.Ortaya çıkan bölge ve komşuları. Gruplandırma süreci en benzer bölgeler, bütün görüntü tek bir bölge haline gelinceye kadar tekrarlanır.

Farklı sahne ve ışık koşullarını hesaba katmak istiyoruz.Aşağıdaki değişkenlik gösteren renk değerleri göz önüne alınır.RGB(red,green,blue),I(gri resim için ,yoğunluk) .

colour channels	R	G	B	I	V	L	a	b	S	r	g	C	H
Light Intensity	-	-	-	-	-	-	+/-	+/-	+	+	+	+	+
Shadows/shading	-	-	-	-	-	-	+/-	+/-	+	+	+	+	+
Highlights	-	-	-	-	-	-	-	-	-	-	-	+/-	+

colour spaces	RGB	I	Lab	rgI	HSV	rgb	C	H
Light Intensity	-	-	+/-	2/3	2/3	+	+	+
Shadows/shading	-	-	+/-	2/3	2/3	+	+	+
Highlights	-	-	-	-	1/3	-	+/-	+

Şekil 27. Seçici Arama Gösterim

En son Lineer Regresyon kullanılarak bu bölgelerdeki nesneler çerçeve içine alınıyor.

R-CoNN ile ilgili Sorunlar ise: görüntüyü başına 2000 bölge teklifini sınıflandırmak zorunda kalacağınız gibi, ağı eğitmek için hala çok zaman gerekir.

Her test görüntüsü için yaklaşık 47 saniye sürdüğü için gerçek zamanlı olarak uygulanamaz.

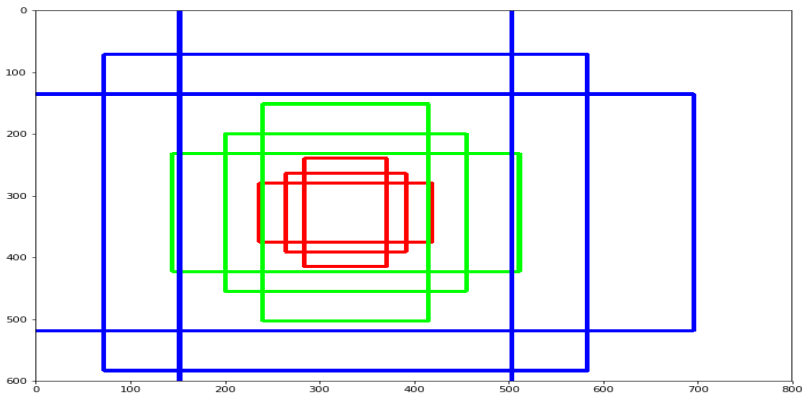
Seçici arama algoritması sabit bir algoritmadır. Bu nedenle, o aşamada hiçbir öğrenme gerçekleşmez. Bu, kötü aday bölgesi tekliflerinin oluşturulmasına yol açabilir.

2.3.3 Hızlı R-CoNN(Faster R-CoNN)

Daha hızlı bir nesne algılama algoritması oluşturmak için R-CoNN'nin bazı dezavantajlarını çözdü ve buna Hızlı R-CoNN adı verildi.Hızlı R-CoNN ile buradaki ana fark, daha sonra bölge önerileri oluşturmak için seçici bir arama kullanmasıdır.[6]

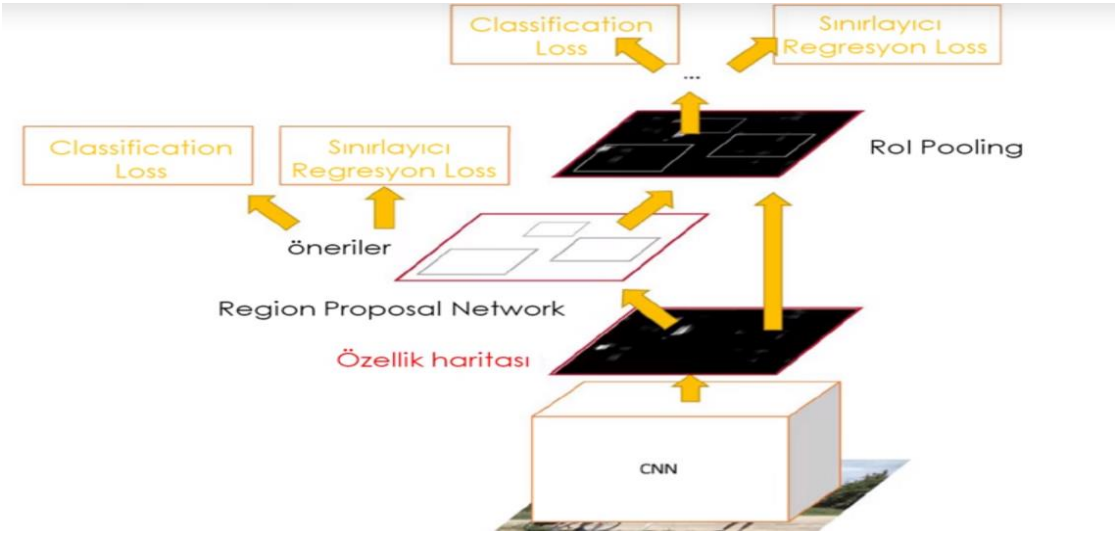
Yaklaşım R-CoNN algoritmasına benzer. Ancak, bölge önerilerini CoNN'ye beslemek yerine, evrişimli bir özellik haritası oluşturmak için girdi görüntüsünü CoNN'ye besleniyor.

Anchors(çapa) önemli rol oynar Faster R-CoNN’in başarısında.Bir çapa bir kutudur. Faster R-CoNN'nin varsayılan konfigürasyonunda, bir görüntünün konumunda 9 çapa vardır. Aşağıdaki grafikte, (600, 800) boyutunda bir görüntünün konumunda (320, 320) 9 çapa gösterilmektedir.[9]

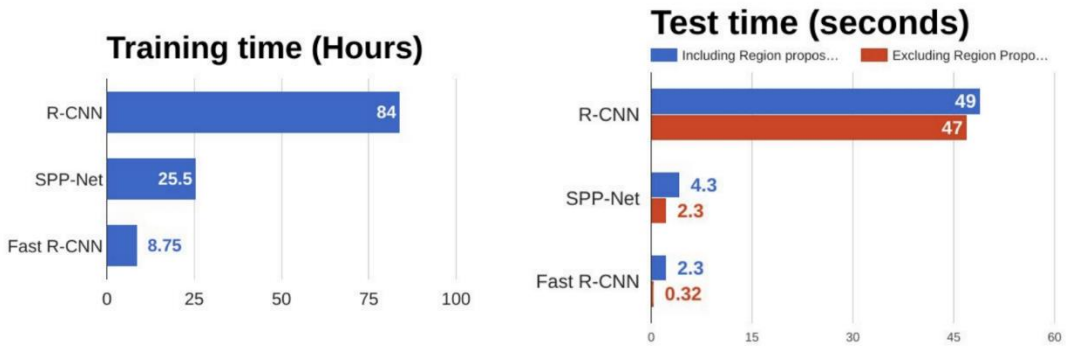


Şekil 28. Çapa Gösterim

Hızlı R-CoNN sebebi R-CoNN'den daha hızlı olmasının sebebi, her seferinde evrimsel sinir ağına 2000 bölge teklifini beslemeniz gerekmemesidir. Bunun yerine, evrişim işlemi görüntü başına sadece bir kez yapılır ve ondan bir özellik haritası oluşturulur.



Şekil 29. Daha Hızlı CoNN



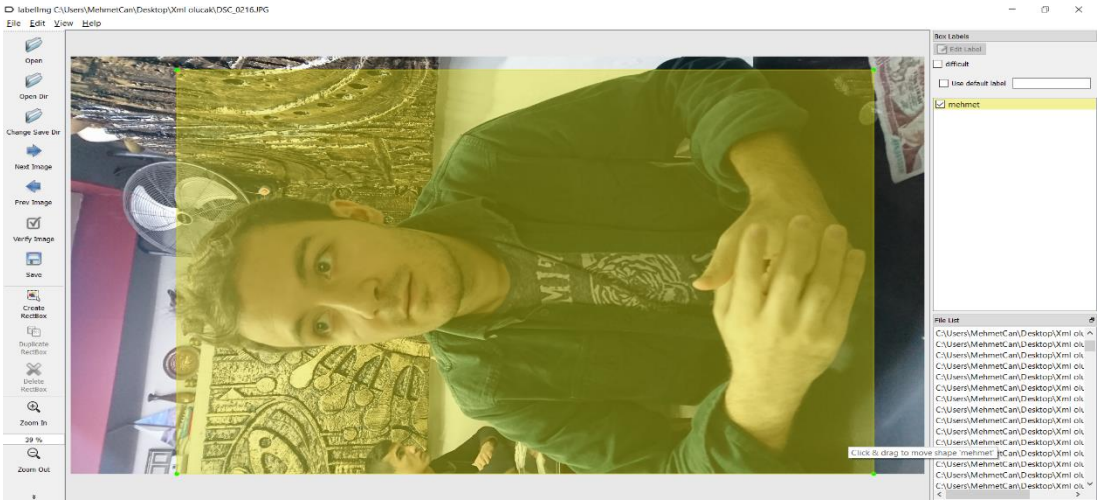
Şekil 30. Daha Hızlı CoNN-1

Yukarıdaki grafiklerden, Fast R-CoNN'nin R-CoNN üzerinden eğitim ve test seanslarında çok daha hızlı olduğunu görebilirsiniz. Fast R-CoNN'in test süresi boyunca performansına baktığınızda, bölge önerileri de dahil olmak üzere, bölge önerileri kullanmamaya kıyasla algoritmayı önemli ölçüde yavaşlatır. Bu nedenle, bölge önerileri performansını etkileyen Hızlı R-CoNN algoritmasında darboğazlar haline geldi.

3. PROJEDE KULLANILAN YÖNTEMLER VE METHODLAR

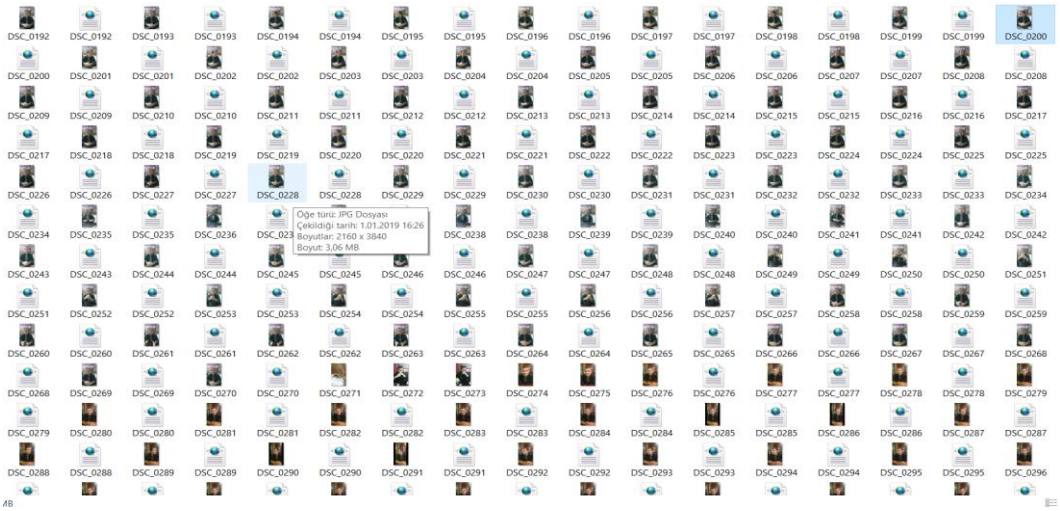
3.1 Kullanılanlar Veritabanı

LabelImg'deki image dizinini açabilir ve sınırlayıcı kutu ve bir etiket adı ekleyebilirsiniz. Kaydettiğinizde, sınırlama kutusunun (xmin, ymin xmax, ymax) ve etiket adının x, y koordinatlarının ayrıntılarını içeren bir .xml dosyası verir..xml dosyalarını oluşturmak için görüntüleri etiketlemektir. Her görüntünün, görüntüdeki nesnenin konumunu belirtmek için sınırlayıcı kutulara ihtiyacı vardır. Bu amaçla LabelImg açık kaynak kodlu bir araç kullandım.[15]



Şekil 31.Etiketleme Gösterim

Bu adımın sonunda, .jpg (data) images ve .xml (label) dosyalarına sahip bir klasörünüz olmalıdır. Örneğin, iki sınıfı tespit etmek için modeli eğitiyorsak .



Şekil 32.Etiketlenip Csv'leri Oluşturulmuş Klasör

3.2 Kullanılanlar Araçlar ve Kütüphaneler

3.2.1 TensorFlow

TensorFlow Google tarafından desteklenen açık kaynak bir frameworktür.Karmaşık graf modellerini hesaplamada kullanılır ve bu modelleri analiz etmeyi kolaylaştırır.Çok boyutlu diziler Tensor diye isimlendirilir ve aynı görevi yaparlar.En önemli özelliklerinden birisi neredeyse her sorun çok iyi şekilde dokümantasyon hazırlanmış olması, geliştirenler tarafından.[17]

TensorFlow uygulama geliştiriciler için dizayn, derlemek ve nöron ağlarını eğitmek.Daha bir çok framework bulunmakta bu işleri yapan fakat TensorFlow bunlar arasında en çok kullanılanı.Keras gibi yüksek kalibreli kütüphaneler arka plan da TensorFlowa ihtiyaç duyar.

TensorFlow platform bağımsızdır.GPU veya CPU , gömülü sistem, mobil gibi bir çok platformda çalıştırılabilir.

TensorFlow deposunda bir çok yayınlanmış Graf modeli kodları dışında eğitilmiş, eğitilmiş model ağırlarınıda içeriyor.Object Detection API ile bu ağırlıkları kendi modellerinizde deneyebiliriz .

TensorFlowun GPU ve CPU üzerine kurulması şeklinde iki seçeneği bulunuyor.GPU normalde görüntülerdeki grafları çözümleyerek işlem yaptığından ve bunlar derin öğrenmedeki işlemlere benzerliğinden dolayı ve paralel işlemler yapmasından dolayı GPU kullanımı tercih edilir.Hali hazırda GPU daha komple bir donanımdır CPU ya göre daha fazla çekirdeği bulunur, daha hızlıdır ve CPU'nun bilgisayarın olağan işleriylede olan meşgullüğü tercihi GPU'ya yöneltir.

Yalnız TensorFlowun GPU'yu kullanması için bazı ek yazılımlara ihtiyaçlara vardır.Nvidia ekran kartları için CUDA ve cuDNN kurulumları bunlar kurulduktan sonra kullanıma hazır olacaktır.

3.2.2 CUDA

Nvidia tarafından geliştirilen GPU'yu daha etkin kullanabilmek için çekirdeklerin paralel hesaplamasına izin veren bir API'dir.CUDA ile geliştiricilerin direk olarak GPU'ya erişimine olanak sağlanır.Bunlar sayesinde ileri seviye görüntü işleme işlemlerinde tercih edilir.

Main Memoryden(RAM) gelen Process verisi,GPU ön belleğine gelir bu sırada CPU , GPU'ya çalışma komutu verir ve GPU da CUDA nın sağladığı özellikle çekirdeklerin paralel işlemleriyle veri işler ve Ram'e sonucu geri döndürür.

3.2.3 cuDNN

cuDNN, Nvidia CUDA'nın derin öğrenme kütüphanesidir.GPU'dan derin öğrenmede normalizasyon,konvülasyon vb. yerlerde daha fazla verim alınmasını sağlar.Bunu doğrudan geliştirici kullanabilir fakat TensorFlow gibi kütüphaneler soyutlayarak onu etkin şekilde kullanabilir.
cuDNN, Caffe2, MATLAB, Microsoft Cognitive Toolkit, TensorFlow, Theano ve PyTorch gibi yaygın olarak kullanılan derin öğrenme çerçevelerini hızlandırır.
GPU'nun CPU'dan daha etkin olduğu ve GPU'nun cuDNN ile performansının daha da artmıştır.

3.2.4 Keras

Derin Öğrenme framework'ü ve Pythonla yazılmıştır.Front-end olarak çalışır, back-end kısmında TensorFlow, Theano tercihe göre kullanılabilir.
Keras daha çok katmanların düzenlemesine yoğunlaşmıştır.Modelleri tanıma ve eğitme konusunda çok kolaylık sağlıyor.CoNN ve RNN 'i destekliyor ve birlikte kullanabilme imkanı sağlıyor.

3.3 Kullanılan Method

3.3.1 Nesne algılama API'sini yükleme

Veritabanımızı düzenledikten sonra. Nesne algılama API'sini yüklüyoruz. Başlamadan önce, nesne algılama API'sini klonlayıp dosyalarımıza yüklemeliyiz.

3.3.2 Etiket dosyalarını (.xml) .csv formatına dönüştürme

Etiket dosyalarını .csv formatına dönüştürmek için ipynb dosyasındaki talimatları izleyin. Komutları doğrudan Jupyter dizüstü bilgisayardaki "çalıştır" düğmesini tıklatarak çalıştırabilir veya ipython'u terminalinizde çalıştırabilir ve komutları ipynb dosyasından kopyalayıp yapıştırabilirsiniz.

3.3.3 Etiketleri ikili biçime dönüştürme (TFRecord)

Büyük veri kümeleriyle çalışıyorsanız, verilerinizin saklanması için ikili bir dosya biçiminin kullanılması, içe aktarma hattınızın performansını ve bunun sonucunda modelinizin eğitim süresini önemli ölçüde etkileyebilir. İkili veri diskte daha az yer kaplar, kopyalanması daha az zaman alır ve diskten çok daha verimli bir şekilde okunabilir.TFRecord, TensorFlows ikili depolama formatıdır. .Test ve eğitim veri

setleri için TFRecord dosyalarını oluşturmak için ipynb dosyasındaki talimatları izlenir.

3.3.4 Yeniden eğitilecek bir model seçimi

Veri setinizle baştan sona bir model eğitimi uzun zaman alabilir (haftalar) ve ileri seviye GPU'lar gerekebilir. Bunun yerine, önceden eğitilmiş bir model kullanabilir ve bu modeldeki sınıfları sınıflarınızla değiştirmek için veri setinizle yeniden eğitebilirsiniz. Bu proje için temel model olarak Mobileset SSD'yi seçtik. SSD modelleri yüksek hız sunar ve video yayınlarında algılama için idealdir. Mobilenet, düşük doğrulukta cihazlar için uygundur, çünkü yine de daha az doğruluk sağlarken daha az yer kaplar. Mevcut modellerin bir listesi tensorflowun github kısmında mevcuttur.

3.3.5 Proje Kodları ve Açıklamaları

Seçtiğim önceden eğitilmiş model Faster RcoNN modelidir.Num_classes kısmı kişi tarafından tanınmasını istediği ve .xml dosyalarını oluşturduğu resimlerdeki etiketlerdeki sınıflardır.Dimension kısmında eğitim için verilen resimlerdeki boyutlandırma sınırlarını belirtiliyor.

```
model {
  faster_rcnn {
    num_classes: 6
    image_resizer {
      keep_aspect_ratio_resizer {
        min_dimension: 600
        max_dimension: 1024
      }
    }
    feature_extractor {
      type: 'faster_rcnn_inception_v2'
      first_stage_features_stride: 16
    }
```

Aşağıdaki şekilde eğitim için oluşturulmuş veri setini etiketlerken oluşturuluan .xml dosyası oluşturulur.Burada size kısmında resmin boyutu verilmiştir. Name kısmında etiketlenen kişi veya nesnenin etiket ismi.Object kısmında etiketlenecek nesnenin resim üzerindeki koordinatları verilmiştr.Bu her resim için tekrar edilir.

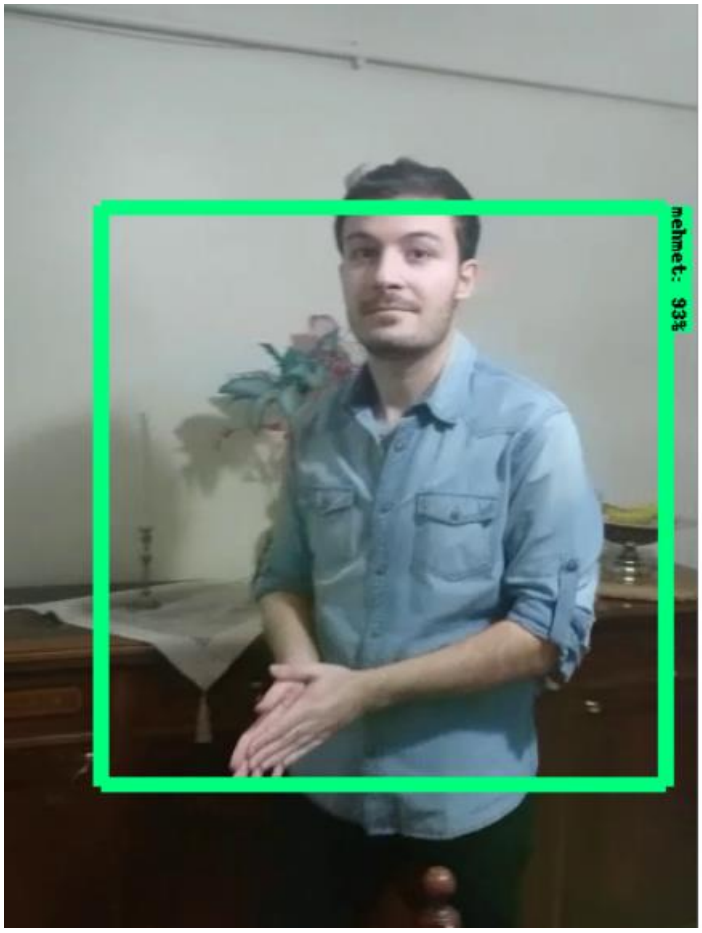
```xml
<annotation>
    <folder>Yeni klasör (2)</folder>
    <filename>DSC_0192.jpg</filename>
    <path>C:\Users\MehmetCan\Desktop\Yeni klasör (2)\DSC_0192.jpg</path>
    <source>
        <database>Unknown</database>
    </source>
    <size>
        <width>304</width>
        <height>540</height>
        <depth>3</depth>
    </size>
    <segmented>0</segmented>
    <object>
        <name>mehmet</name>
        <pose>Unspecified</pose>
        <truncated>0</truncated>
        <difficult>0</difficult>
        <bndbox>
            <xmin>11</xmin>
            <ymin>86</ymin>
            <xmax>300</xmax>
            <ymax>501</ymax>
        </bndbox>
    </object>
</annotation>
```

Şekil 33.xml dosyanın içerisi

```
train_config: {
  batch_size: 1
  optimizer {
    momentum_optimizer: {
      learning_rate: {
        manual_step_learning_rate {
          initial_learning_rate: 0.0002
          schedule {
            step: 0
            learning_rate: .0002
          }
          schedule {
            step: 900000
            learning_rate: .00002
          }
          schedule {
            step: 1200000
            learning_rate: .000002
          }
        }
      }
      momentum_optimizer_value: 0.9
    }
    use_moving_average: false
  }
```

Burada optimizasyonda kullandığım momentum algoritmasının gradient-descenti bulurken hangi değerler üzerinden hareket etmesi gerektiği belirtildi.

En son aşamada eğitilmiş modelimizi test etmek için resim,video veya web cam kullanılarak test verisi verilebilir.Ben video ve web cam olarak örnekledim.Videonun bir kesitindeki resim aşağıdaki şekilde beni, öncesinde etiketlediğim şekilde bulabiliyor.



Şekil 34.Test verisinin sonucu

4. SONUÇ VE ÖNERİ

Bu çalışmada nesne belirleme ve tanıma için Tensorflow kullanılmıştır.Derin öğrenme algoritması olarak CoNN ve kodlanmasında python dili kullanılmıştır.Nesne tanıma sürecinin dolayı tensorflow modelinin tercih edilme nedeni görüntü işleme konusunda donanımsal olarak eğer yeterli ekipman sahibi değilseniz.Dışarıdan imkanları daha fazla olan kurumlar tarafından eğitilmiş modelleri kullanmak gerekiyor.Modelin ikinci defa eğitilme süreci benim tarafımdan ekran kartı üzerinden yapılmıştır(GTX 960m) ve böyle yapılması tavsiye edilir. Çünkü normalde görüntüler üzerine işlemler için tasarlanan ekran kartları aynı şekilde görüntü üzerinde özellik çıkarımı yapılacağı zamanda işlemciye göre daha fazla verim vermektedir.Ben veri olarak kendi resimlerimi kullandım sonucunda modelin beni tanımasını sağlayacak şekilde düzenledim ve gerçekleşti.Burada dikkat edilmesi gereken kısım veri setinizdeki resimlerin boyutları, eğer çok yüksek özellikli bir sisteminiz yoksa buradaki resimlerin boyutlarını küçük tutmanızı tavsiye ederim yoksa benim ilk başlarken karşılaştığım belleğin dolması ve şeklinde hatalar olacaktır. Nesne belirleme ve tanımada kullanılan CoNN algoritması çok iyi sonuçlar vermekte ama donanımsal olarak yeterli düzeyde ürünlerimiz yok bunları kapatmak için matematiksel kısma yöneliyoruz ve oradan performansı arttırmaya çalışıyoruz.İleride daha iyi performanslı ürünler geldikçe bu alandaki sonuçlar son zamanlardaki artışında üzerine çıkıp bir çok alanda kullanılabileceğini düşünüyorum.

ÖZGEÇMİŞ

1994 yılında İzmit'te doğdum.İlk, orta ve lise eğitimini Kocaeli'nde tamamladım.2013 yılında girdiğim lisans yerleştirme sınavında aldığım puanla Kocaeli Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliğine yerleştim.

İletişim:

E-mail: mehmetcan.akay4@gmail.com

GSM: 0538 796 59 83

KAYNAKÇA

[1]. A. Krizhevsky, I. Sutskever, and G. Hinton. ImageNet classification with deep convolutional neural networks. In NIPS,2012. 1, 4, 6

[2]. Karen Simonyan & Andrew Zisserman- VERY DEEP CONVOLUTIONAL NETWORKS FOR LARGE-SCALE IMAGE RECOGNITION arXiv technical report, September 2014

[3]. Kaiming He, Xiangyu Zhang, Shaoqing Ren, Jian Sun- Deep Residual Learning for Image Recognition 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[4]. Min Lin, Qiang Chen, and Shuicheng Yan. Network in network. CoRR, abs/1312.4400, 2013

[5]. R. Girshick, J. Donahue, T. Darrell, and J. Malik. Regionbased convolutional networks for accurate object detection and segmentation. TPAMI, 2015. 5, 7, 8

[6]. Ross Girshick- Fast R-CNN In ECCV, 2015

[7]. Christian Szegedy, Wei Liu, Yangqing Jia, Pierre Sermanet- Going deeper with convolutions In CVPR, 2014.3

[8]. Shaoqing Ren, Kaiming He, Ross Girshick, and Jian Sun- Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks-IEEE Transactions on Pattern Analysis and Machine Intelligence June 2015

[9]. You Only Look Once: Unified, Real-Time Object Detection Joseph Redmon , Santosh Divvala, Ross Girshick , Ali Farhadi Conference: 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)

[10]. Matthew D. Zeiler and Rob Fergus- Visualizing and Understanding Convolutional Networks Computer Vision – ECCV 2014: 13th European Conference

[11].http://ibrahimdelibasoglu.blogspot.com/2017/10/python-selective-search-segmentation.html

[12].https://www.kdnuggets.com/2016/09/beginners-guide-understanding-convolutional-neural-networks-part-1.html

[13]. https://www.coursera.org/learn/convolutional-neural-networks

[14].http://cs231n.stanford.edu/slides/2017/cs231n_2017_lecture11.pdf

[15].https://blog.goodaudience.com/food-detection-app-using-tensorflow-object-detection-apis-1b9302a9aad2

[16]. Efe, Abadoğlu, E., & Kaynak, O. (1999). Analysis and Desing of a Neural Network Assisted Nonlinear Controller for a Bioreactor. International Journal of Robust and Nonlinear Control, 9(11), 799-815.

[17].https://en.wikipedia.org/wiki/TensorFlow

[18] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner. Gradient-based learning applied to document recognition.*Proceedings of the* IEEE, november 1998.