T.C.
SAKARYA ÜNİVERSİTESİ
BİLGİSAYAR VE BİLİŞİM BİLİMLERİ FAKÜLTESİ

BSM 401 BİLGİSAYAR MÜHENDİSLİĞİ TASARIMI

TENSORFLOW VE OPENCV KÜTÜPHANESİ İLE ANDROİD PLATFORMUNDA NESNE TESPİTİ

B161210074 -Ahmet Yasir AKBAL

Bölüm	**:**	**BİLGİSAYAR MÜHENDİSLİĞİ**
Danışman	**:**	**Dr.Öğr.Üyesi Muhammed Fatih ADAK**

2019-2020 Güz Dönemi

ÖNSÖZ

Günümüzde Yapay Zekanın, sağlık, iletişim, güvenlik, otomotiv gibi bir çok sektörde kullanımı gün geçtikçe yaygınlaşmaktadır. Bir çok şirket ve devlet yatırımlarını bu alanda arttıkmakta, çoğu üniversitede bu ders zorunlu olarak verilmektedir. Bu raporda Bulanık Mantık, Yapay Sinir Ağları/Derin Öğrenme, Makine Öğrenmesi, Genetik Algoritmalar, Uzman Sistemler gibi alt başlıklara ayrılan Yapay Zekanın, Yapay Sinir Ağları/Derin Öğrenme alt başlığı üstünde yoğunlaşacağız. Bu alt alan uzun süredir varolmakla birlikte son günlerde yaşanan teknolojik gelişmeler sonucu donanım maliyetinin azalması, veri miktarının artması, donanım performansının artması ve yeni oluşturulan daha iyi sonuçlar veren yapay sinir ağı modelleri ile patlama yaşamıştır. Önümüzdeki dönemde de bu gelişimin devam etmesi beklenmektedir.

İÇİNDEKİLER

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Yapay Zeka, Görüntü İşleme, Bilgisayarlı Görü, Mobil Uygulama

Yapay Zeka, bilgisayarlara insanların yaptığı gibi bir olayı algılayıp, bu olay üzerinde düşünüp sonuç çıkarma kabiliyetinin kazandırılmasını hedefler. Burada yapay zekanın algıladığı şey yapay zekanın üzerinde işlem yapacağı veridir. Bu veri görüntü, ses veya metin olabilir. Yapay zeka aldığı bu veriyi işleyerek bir sonuca varır. Örnek olarak en optimal yolun bulunması, resimdeki nesnenin tespit edilmesi, resmin sınıflandırılması verilebilir. Bu kabiliyet ona yine insanlar tarafından kazandırılır. Burada önemli olan belli veriler kullanılarak oluşturulan bu sistemin algılama/sonuç çıkarma kabiliyetinin genelleştirelerek daha önce sistemin görmemiş olduğu veriler üzerinde de uygulunabilmesidir. Bu genelleştirme işleminin kapsamı bazı parametrelere dayalı olarak değişebilir.

Görüntü işleme adından da anlaşıldığı gibi görüntüler üzerinde işlem yapılmasıdır. Görüntüler üzerinde yapılacak işlemler çok çeşitlidir, örnek olarak resmin boyutunu değiştirme, renkleri ters çevirme, bulanıklaştırma verilebilir. Bilgisayarlı görü ise insanın gözüyle yaptığı algılamaları makinelere yaptırmaktır. Eskiden yapay zekanın bir alt dalı olarak görülse de günümüzde başlı başına bir alan haline gelmiştir. Bilgisayarlı görüde de yapay zeka teknikleri kullanılır.

Tasarladığım yazılımda görüntü işleme, bilgisayarlı görü ve yapay zeka kullanılarak Java dilinde bir Android uygulaması geliştirilmiştir. Yapay zeka kütüphanesi olarak Tensorflow, görüntü işleme kütüphanesi olarak OpenCV kullanılmıştır.

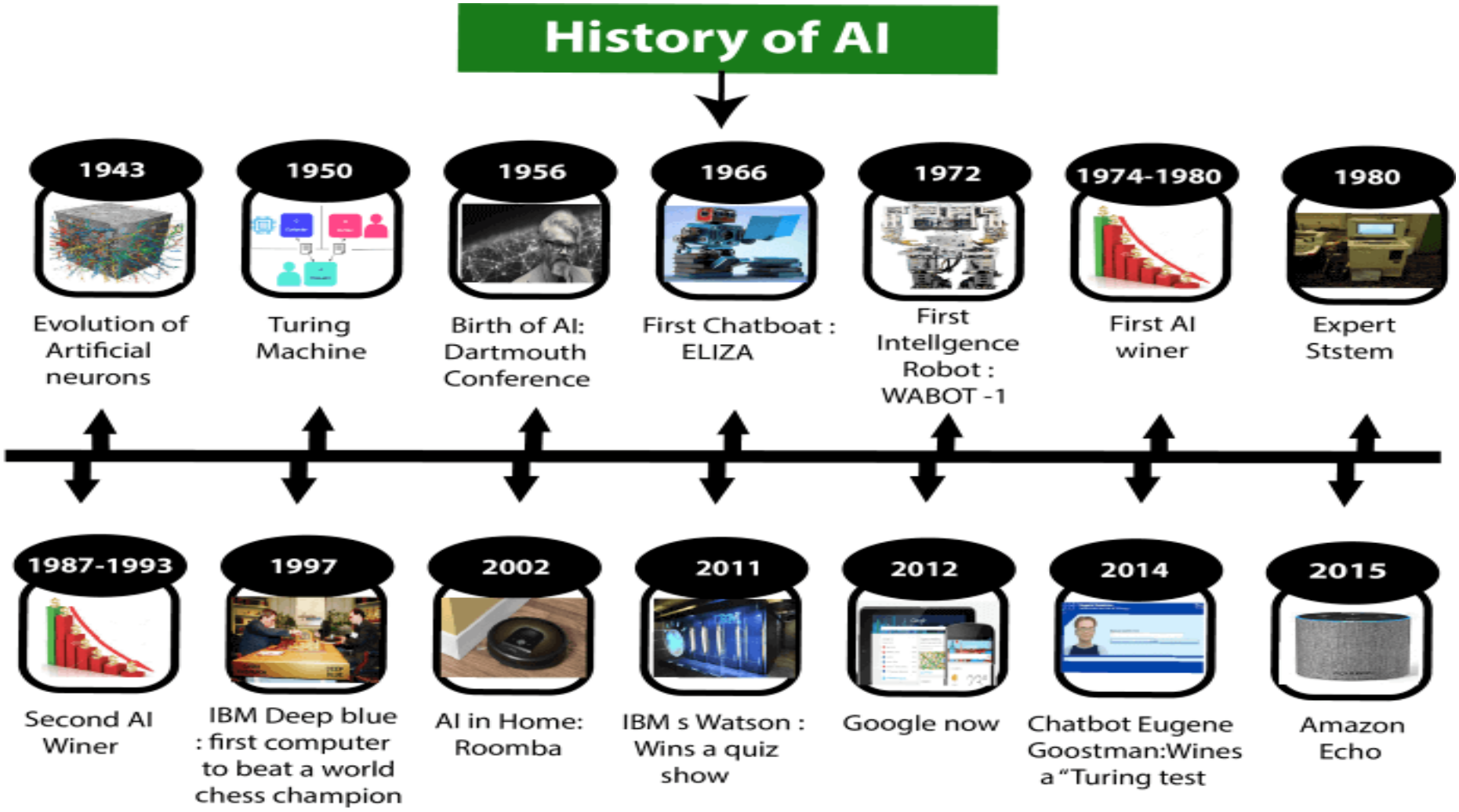
BÖLÜM 1. GİRİŞ

Yapay zekanın tarihi milattan önceye kadar gitmektedir. Antik Yunan döneminde insanımsı robotlar ile ilgili çeşitli fikirlerin ortaya atıldığı bilinmektedir. Bu yazıda modern yapay zekadan bahsedilecektir. Modern yapay zekadan kasıt bilgisayarlara zeka özelliğinin kazandırılmasıdır.

1.1. Yapay Zekanın Tarihçesi

Yapay zeka için 1884 yılı oldukça önemlidir. Bu tarihte Charles Babbage, zeki davranışlar gösterecek bir mekanik makine üzerinde çalışma yapmıştır. Fakat çalışmaları sonucu insanlar kadar zeki davranışlar sergileyecek bir makine üretilemeyeceğine karar vermiş ve çalışmalarına ara vermiştir. 1950 yılında Alan Turing bir makinenin zeki olmadığına karar veren meşhur Turing Testi'ni oluşturmuştur. O dönemde bu testi geçebilen makinelerin zeka seviyesi yeterli sayılmaktaydı. 1956 yılında yapay zeka resmi olarak ilk kez İngilteredeki bir konferansda John McCarty öncülüğünde ortaya atılmıştır. Daha sonra 1957 yılında John McCarty yapay zeka uygulamaları için fonksiyonel LISP(List Processing Language) dilini geliştirmiştir.

1965-1970 yılları arası yapay zeka için karanlık bir dönem olarak adlandırabilir. Ortaya çıkan gerçekçi olmayan beklentiler nedeniyle oluşan aceleci ve iyimser tutum insanlar kadar zeki olabilecek makinelerin üretilmesinin kolay olacağı düşüncesini oluşturmuştur. 1970-1975 yılları arası ise yapay zekanın ivme kazandığı bir dönemdir. 1975-1980 yılları arasında ise psikoloji başta olmak üzere başka bilim dallarından yapay zekaya katkı sağlanabileceği düşüncesi oluşmuştur. 1980 yıllardan günümüze kadar ise yapay zeka gelişimine devam etmiştir. Günümüzden örnek olarak sürücüsüz araçlar, nesne tespiti, ses tanıma, yüz tanıma, doğal dil işleme verilebilir.



Şekil 1.1. Yapay zeka tarihçesi

1.2. Yapay Zeka

Yapay zekadan bahsetmeden önce zeka ve aklın tanımını yapalım. Zeka, gerçekleri algılama, yargılama ve sonuç çıkarma yeteneklerinin tamamıdır. Akıl ise düşünme, anlama, kavrama, idrak etme, karar verme ve önlem alma yetenekleridir[1]. Akıl, genetik yoldan intikal eden sevgi, korku, kıskançlık, doğal savunma güdülerinin yanı sıra hayat deyimlerimiz, çevremizden aldığımız etkileşimlere bağlı olarak değişmektedir. Yani akıl yaşamamız boyunca değişim gösterebilir. Akıl etik bir anlam taşır diyebiliriz. İşin içine duygu da girer ve akıl herhangi bir makine, yazılım ile taklit edilemez.

Zeka ise etik bir anlam taşımaz. Akıldan farklı olarak net olarak ölçülebilir. Buna İQ testini örnek verebiliriz. Zeka da akıl gibi doğuştan gelen etmenlere bağlıdır. Öğretilerek, eğitilerek, edilinen bilgi ve birikimlerle geliştirebilir. Zeka ile akıl arasındaki farklı bir cümleyle şöyle de açıklayabiliriz ; zeki bir insan bir probleme iyi ve hızlı bir çözüm üretebilir, akıllı insan ise ürettiği çözümü olumlu yönde kullanma kabiliyetine sahiptir. Zeka bir makine veya yazılım tarafından taklit edilebilir. Biz bunu yapay zeka olarak tanımlıyoruz. Yani yapay zekanın amacı bir makineye zeka niteliğinin kazandırılmasıdır. Yapay zeka bir çok alt alana ayrılmaktadır. Biz bunlara alt başlıklar altında genel olarak değineceğiz.

1.2.1. Bulanık mantık

Bilgisayarlarda geleneksel yaklaşımı yani 0/1 mantığını kullanırız. Bu yaklaşımda olay ya gerçekleşir(1 değerini alır) ya da hiç gerçekleşmez(0 değerini alır). Örnek olarak bir su ya sıcaktır ya da soğuktur, bir insan ya uzundur ya kısadır. Tabi bunlara daha çeşitli sınıflandırmalar eklenebilir. Bir su sıcak, az sıcak, çok sıcak olabilir, bir insan kısa ,orta, uzun boylu olabilir. Genel mantıksal yaklaşımda bunlar mümkündür fakat bazı problemler vardır. Geleneksel yaklaşımda 165 ile 175 arasındaki insanları orta boylu olarak belirleyelim. Bilgisayar için 165 boyundaki bir insan da 174 boyundaki insan da aynı oranda orta boyludur. Yani bu aralıkta her insanı aynı

şekilde sınıflandırır. Hepsi orta boyludur, bilgisayar her biri için 1 değerini üretir. Bu aralık dışındakiler için de 0 değerini üretir.

Biz bu yaklaşımla bir arabanın kalitesini motor gücü, maksimum hız, dayanıklılık, motor ömrü gibi parametrelere göre bulmak isteyelim. Her bir parametre için sınıflandırma yaparız. Motor gücü, az güçlü, orta güçlü, ortanın üstü güçlü, güçlü, çok güçlü vb. Geleneksel yaklaşımda ne kadar hassas bir hesaplama yapmak istiyorsak parametre sayımız da o oranda fazla olmalıdır. Fakat bizim her aralık için farklı değerler almak için çok fazla miktarda özellik tanımlamamız gerekir bu geleneksel yaklaşımın sorunudur. Bu yaklaşımla ile bilgisayarlara insanlar gibi sınıflandırma yapma kabiliyeti kazandıramayız.

Bulanık mantıkta ise bir özellik bir nesnede ya vardır ya yoktur şartı yoktur. Yani 0/1 mantığı yoktur, onun yerine bir özellik bir nesneye 0-1 aralığı arasında bir üyelik derecesi ile sahiptir. Örneğin 175 cm boyundaki bir insan 0.6 üyelik derecesi ile orta boylu, 0.3 üyelik derecesi ile de uzun boylu hatta bunun yanıda 0.1 üyelik derecesi ile çok uzun boylu olabilir. Üyelik dereceleri üyelik fonksiyonlarından belirlenir. Üyelik fonksiyonlarında üçgen, yamuk, gauss eğrisi gibi geometrik şekiller bulunur. Bu şekillerin x ekseni sınırları ile parametrelere tanımladığımız özelliklerin sınırları belirlenir. Y ekseni değeri ise bize üyelik x noktası için üyelik derecesini verir. Üyelik fonksiyonlarında şekillerin kesişmesi ile de bahsettiğimiz bulanıklık sağlanır. Biz belirlenen yönteme göre bu kesişen özelliklerden birini çözüme dahil ederiz. Bu şekilde bir sistem belli parametreler ile bir sonuca varırken daha hassas hesaplamalar yapabilir ve klasik küme mantığındaki sorunlar giderilerek bilgisayarların 0/1 mantığı ile değil de insanlar gibi sınıflandırma yapabilmesi sağlanmış olur.

1.2.2. Genetik algoritmalar

Genetik algoritmalar yapay zekanın gittikçe genişleyen bir kolu olan evrimsel hesaplama tekniğinin bir parçasını oluşturmaktadır. Genetik algoritma Darwn'in evrim kuramı olan doğada en iyinin yaşaması kuralından esinlerek oluşturulan, bir veri öbeğinden özel bir veryi bulmak için kullanılan bir arama yöntemidir. Genetik

algoritma geleneksel yöntemlerle çözümü zor veya imkansız olan problemlerin çözümünde kullanılmaktadır.[2]

1.2.3. Makine öğrenmesi

1959 yılında Arthur Samuel makine öğrenimini: “makinelerin bilhassa programlanmadığı sonuçları öğrenebilme kabiliyeti” olarak tanımlamıştır. Makine öğreniminin kullanım yaygınlığı veri madenciliğinin ortaya çıkması ile 1990’lı yıllardan sonra oldukça artmıştır.[3]

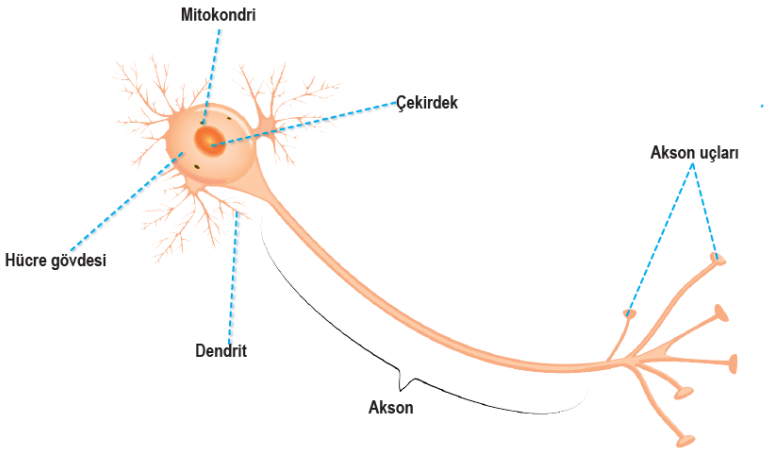
Makine öğrenmesi bir çok kişi tarafından yapay zekanın alt dalı olarak kabul edilir. Yapay zekanın hedefi makinelere zeka niteliğinin kazandırılmasıdır. Bu özellik makineye 3 şekilde verilebilinir. İlkinde klasik programlama teknikleri ile bir programa ne yapacağı harfi harfine verilir. Program bu söylenen kurallar dışında bir şey yapmaz. Bunu zayıf yapay zeka olarak adlandırabiliriz. Yine de veri ve parametre sayımız az ise bu yöntem ile bir makineye zeka özelliğini kazandırıp bir insanın arkada işlemleri yapanın bir insan mı yoksa makine mi olduğunu ayırt edemeyecek seviyeye getirebiliriz. Fakat veri ve parametre sayısı arttıkça bu işlem gittikçe imkansızlaşır. İşte bu noktada makine öğrenmesi işleri kolaylaştırır. Makine öğrenmesinde de kural tanımı vardır. Bu kurallar insanoğlunun tecrübeleri ile gelen bilgiler sonucu oluşturulur. Makine öğrenmesi kullanan sistemler tanımlanan kuralları harfi harfine uygulayarak işlem yapmaz, verilen kuralları kullanarak daha önce görmediği veriler üzerinde de tahmin yapabilir. Doğru ve optimal çözümü bulmak için kendini eğitebilir. Bahsettiğimiz makineye zeka özelliğinin kazandırılması için kullanılan metotlardan sonuncusu olan derin öğrenmeye bir sonraki alt başlıkta değineceğim.

1.2.4. Derin öğrenme/yapay sinir ağları

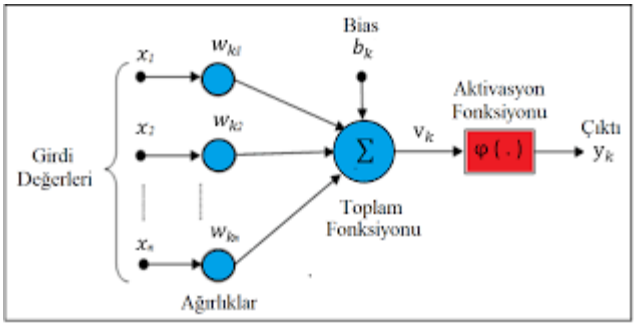
Derin öğrenme ve yapay sinir ağlarını makine öğrenmesinin bir alt dalı olarak görebiliriz. Önceki bölümde veri ve parametre sayısı arttıkça zayıf yapay zeka teknikleri ile bir makineye zeka niteliğinin kazandırılmasının imkansızlaştığını söylemiştik. Buna çözüm olan makine öğrenmesinden bahsetmiştik. Veri ve

parametre sayısı daha da arttığında makine öğrenmesi ile makineye zeka niteliğinin kazandırılması da zorlaşmaktadır. Bu noktada derin öğrenme/yapay sinir ağları bu soruna çözüm olmaktadır. Makine öğrenmesinde olduğu gibi insanoğlunun bilgi ve deneyimlerine ve bunların kurallaştırmasına gerek yoktur. Sistemi eğitmek için sisteme sadece girdiler ve/veya çıktılar verilir. Sistem kendi kendini eğitebilir. Sırasıyla yapay sinir ağları, öğrenme, derin öğrenmeden bahsedeceğim. Son olarak yapay sinir ağları ile derin öğrenmenin ilişkisinden/farklarından bahsedeceğim.

1890 yılında William James nöral öğrenme sürecini tanımlamıştır. Bu sürecin matematiksel model olarak tanımlanması ve üzerinde denemeler yapılması ancak 1943 yılından sonradır. Yapay sinir ağı insanın öğrenme mekanizmasını matematiksel olarak modellenmesi amaçlanarak tasarlanmıştır. İnsanda bu olay nöronlar vasıtası ile gerçekleşir dolayısıyla burada modellenen yapı nöronlardan oluşmaktadır. Nöronlar dentritler, sinaps, hücre çekirdeği ve aksondan oluşur. Nöronların matematiksel olarak modellenmiş halinde dentritleri veri girişinin alındığı elemanlar olarak nitelendirebiliriz.



Şekil 1.2. Nöronun Biyolojik Yapısı



Şekil 1.3. Yapay Nöronun Yapısı

Sinapslar ise her bir veri giriş ucu yani dentrit için ağırlık değeridir. Dentritlerden gelen veri bu ağırlık değeri ile çarpılır. Bulunan bu değerler toplama işlevine verilir.

Toplam $\text{Net} = \sum_{i=1}^{N} X_i * W_i$	Ağırlık değerleri girdiler ile çarpılır ve bulunan değerler birbirleriyle toplanarak Net girdi hesaplanır.
Çarpım $\text{Net} = \prod_{i=1}^{N} X_i * W_i$	Ağırlık değerleri girdiler ile çarpılır ve daha sonra bulunan değerler birbirleriyle çarpılarak Net Girdi Hesaplanır.
Maksimum $\text{Net} = \text{Max}(X_i * W_i)$	n adet girdi içinden ağırlıklar girdilerle çarpıldıktan sonra içlerinden en büyüğü Net girdi olarak kabul edilir.
Minimum $\text{Net} = \text{Min}(X_i * W_i)$	n adet girdi içinden ağırlıklar girdilerle çarpıldıktan sonra içlerinden en küçüğü Net girdi olarak kabul edilir.
Çoğunluk $\text{Net} = \sum_{i=1}^{N} Sgn(X_i * W_i)$	n adet girdi içinden girdilerle ağırlıklar çarpıldıktan sonra pozitif ile negatif olanların sayısı bulunur. Büyük olan sayı hücrenin net girdisi olarak kabul edilir.
Kumilatif Toplam $\text{Net} = \text{Net(eski)} + \sum_{i=1}^{N} X_i * W_i$	Hücreye gelen bilgiler ağırlıklı olarak toplanır. Daha önce hücreye gelen bilgilere yeni hesaplanan girdi değerleri eklenerek hücrenin net girdisi hesaplanır.

Şekil 1.4. Bazı Toplama İşlevleri

Toplama işlevinde kullanılan işleve göre farklı işlemler yapılır. Gelen tüm değerler toplanabilir, verilerin maksimum veya minimum değeri alınabilir, çoğunluk ve normalleştirme algoritmaları kullanılabilir. Genel olarak toplama işlevinde toplama işlemi yapılır. Toplama işleminden çıkan sonuç aktivasyon fonksiyonuna verilir. Aktivasyon fonksiyonunu nöron çekirdeğinin matematiksel karşılığı olarak düşünebiliriz. Aktivasyon fonksiyonları farklı şekillerde olabilir. Aktivasyon fonksiyonuna gelen değer fonksiyona girdi olur, fonksiyondan çıkan değer ise bizim nöronumuzun çıktısıdır. Bu çıktı aksonlar aracılığıyla diğer nöronlara iletilir.

Şuana kadar bahsettiğimiz yapay sinir ağları girdi ve çıktı katmanlarından oluşan 2 katmanlı bir yapıya sahipti. Bu yapı ancak basit ve lineer problemlere çözüm getirebiliyordu. Kullanımı çok yaygın değildi. Dönemin şartlarında yeni modeller oluşturulsaydı bile bu modelleri eğitebilecek donanım gücü ve veri yeterli değildi. Gelişen teknolojinin sonucu olarak veri sayısının ve donanım gücünün artması ile birlikte yeni yapay sinir ağı modelleri geliştirilmesinin yolu açıldı. Sadece girdi ve çıktı katmanlarından oluşan modellere ara katmanlar eklenerek derin sinir ağları(DNN) oluşturuldu. Derin sinir ağlarını kullanarak makine öğrenmesinin gerçekleştirilmesine Derin Öğrenme denir. Derin öğrenme finans, hesaplamalı

biyoloji, enerji üretimi, üretim sektörü, doğal dil işleme gibi alanlarda kullanılmaktadır. Fakat derin öğrenmeyi günümüzde bu kadar popüler yapan görüntü işleme ve bilgisayarlı görüdür. Derin öğrenme ile bu alanda bir resimdeki objeyi konumuyla birlikte tespit edebilir, resimleri sınıflandırabilir, resimleri analiz edebilir, yüz tanıma yapabiliriz. Normal bir yapay sinir ağı modelinde sadece girdi ve çıktı katmanının olduğunu derin sinir ağlarında ise bunlara ilave ara(gizli) katmanların bulunduğunu söylemiştim. Bu ara katmanlar farklı sayılarda, farklı nöron sayılarında ve farklı türlerde olabilir. Bazı katman türleri aşağıda listelenmiştir.

- Fully Connected Layers
- Convolutional Layers
- Pooling Layers
- Recurrent Layers
- Normalization Layeres

Fully Connected Layerların katmanlarında bulunan nöronların her birinden sonraki katmandaki nöronlara bağlantı vardır. Kullanım alanları çok çeşitlidir. Hemen hemen her yapay sinir ağında özellikle çıkış katmanı olarak kullanılır. Konvolüsyonel katmanlarda sinir ağları bulunmaz, bu katmanlarda resim matrisleri üzerinde bir veya birden sayıda filtre ile konvolüsyon işlemi yapılır. Bu katman türü ve birazdan bahsedeceğim pooling katmanları bilgisayarlı görüde çokça kullanıldığı için daha sonra detaylıca açıklamaya çalışacağım.

Havuzlama katmanlarında(pooling layers) önceki katmandan girdi olarak gelen verilere filtreleme uygulanarak işlenecek verinin sayısının düşürülmesi amaçlanmaktadır. Recurrent Layers'lar daha çok daha çok doğal dil işleme, konuşma tanıma, diller arası çeviri gibi alanlarda kullanılır. Normalizasyon katmanlarında ise veriler normalize edilir. Bazı ağlarda bu işlem isabet oranını arttırabilir.

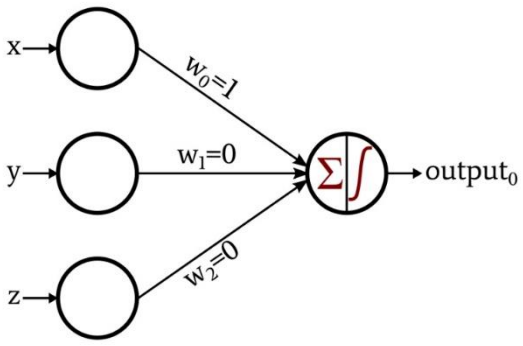
1.2.4.1 Bir yapay sinir ağı nasıl eğitilir ?

Eğitimin nasıl gerçekleştiğini açıklamadan önce eğitim türlerinden bahsedeceğim. 3 çeşit öğrenme türü vardır: Destekli öğrenme, desteksiz öğrenme ve takviyeli

öğrenme. Destekli öğrenmede sinir ağına veri olarak girdiler ile çıktıları da veririz. Bu sayede sinir ağı kendi bulduğu çıktı ile bizim sisteme verdiğimiz beklenen çıktı arasındaki hata payını hesaplayıp doğru yolda olup olmadığını anlayabilir. Kullanım alanı yaygın olmakla birlikte örnek olarak bir resimde nesne tespiti yaparken bu öğrenme çeşidini kullanırız. Benim ödevimde kullandığım öğrenme çeşiti de buydu.

Desteksiz öğrenmede ise sisteme sadece girdi parametreleri verilir. Sistem o veriler için çıktının ne olduğunu bilmez. Verilen tüm veriler için istatiksel dağılıma göre gruplandırma yapar. Bir grup veriyi benzer özelliklerine göre sınıflara bölmek istiyorsak bu öğrenme çeşidini kullanabiliriz. Yapay sinir ağı verileri sınıflandırsa bile biz sisteme çıktı girişi yapmadığımız için sınıflandırdığı verilerin ne olduğunu bilemez. Resim sınıflandırma işlemi yapılırken bu yöntem kullanılır. Takviyeli öğrenmede ise sisteme veri girişi olarak girdileri veririz. Sistem bu girdilere göre bir çıktı üretir. Öğretici bu sonuça bakarak sinir ağının ürettiği çıktının doğru olup olmadığını belirten bir sinyal üretir. Bir diğer yöntem de istenilen sonuça yaklaşılıp yaklaşılmadığını belirten bir sinyal üretmektir. Öğretici makine veya insan olabilir.

Şimdi bir yapay sinir ağı nasıl öğrenir onu açıklamaya çalışacağım. Çok farklı çeşitlerde, farklı amaçlara hizmete eden yaygın yapay sinir ağı tasarımları vardır. Ben burada bir yapay sinir ağının nasıl eğitileceğini anlatmaya çalışırken daha basit olması açısından en ilkel yapay sinir ağı modellerinden biri olan perceptron üzerinden anlatım yapacağım.



Şekil 1.5. Ara katmanı bulunmayan bir perceptron

Yukarıdaki şekilde ara katmanı olmayan bir perceptron modeli örneği verilmiştir. Bu modelde x y z olarak adlandırılan nöronlar girdi katmanında bulunur. Bu nöronlara

dışarıdan sinyal girişi olur. Her nörona gelen veri o nöronun çıkışlarında bulunan kollardaki ağırlık değerleri ile çarpılır. Çıkan sonuç daha önce de bahsettiğim toplama işlevine daha sonra da aktivasyon fonksiyonuna verilir. Aktivasyon fonksiyonundan çıkan değer çıktı değeridir. Ağırlık değerleri eğitimin başlangıcında rastgele olarak verilir. Yukarıda bahsettiğim olaylar sürekli tekrarlanarak yapay sinir ağının bir öğrenme algoritması ile ağırlık değerleri sürekli değiştirilerek bizim eğitim veri setinde istenilen çıktıları vermesi sağlanır. Yani burada eğitimden kasıt ağırlık değerlerinin bizim istediğimiz sonuçlara en yakın sonuçları verecek şekilde değiştirilmesidir. Öğrenmenin daha temeline inecek olursak orada matematik ve algoritmalar vardır. Ben bu raporda bunlardan bahsetme gereği duymadığım için bu kısmı burada keseceğim.

BÖLÜM 2. GÖRÜNTÜ İŞLEME VE BİLGİSAYARLI GÖRME

Görüntü işleme bir görüntünün iyileştirilmesi, sıkıştırılması, bulanıklaştırılması, şekiller eklenmesi, renk uzayının değiştirilmesi, yeniden boyutlandırılması, elde edilen verinin arka plandaki verilerlerle eşleştirilmesi gibi işlemlerin yapıldığı bir alandır. Görüntü işleme ve bilgisayarlı görme aynı şey değildir.

Bilgisayarlı görme eskiden yapay zekanın alt bir dalı olsa da günümüzde başlı başına bir alan haline gelmiştir. Bilgisayarlı görme, gözümüzle yaptığımız algılama biçimini bilgisayara yaptırma amacı taşımaktadır.

Görüntü işleme kavramının üst kategorisi olarak nitelendirebileceğimiz bilgisayarlı görme, günümüzde pek çok alanda kullanılmaktadır. Plaka tanıma sistemleri, yüz tanıma sistemleri bilgisayarlı görme teknolojisi ile gerçekleştirir. İnsanlar çoğu zaman bu kavramı görüntü işleme ile karıştırmaktadır.[4]

Bu kısımdan sonra anlatacaklarım direkt olarak yazdığım yazılımın kodları ile ilgili olduğundan dolayı teorik bilgiler yanında yazdığım kodların bir kısmını da paylaşacak var açıklamasını yapacağım.

2.1. Görüntü ve Görüntü İşleme

Görüntüleri biz insanlar gözlerimiz ile algılarız. İnsanlar için görüntünün tanımını yapmak yersizdir. Burada önemli olan bizim gözümüzle görüp tanıdığımız, algılayabildiğimiz görüntülerin bilgisayarların nasıl algılayabileceğidir. Bilgisayar için görüntü bir veya daha fazla katmanı olan bir matristir. Katman sayısı kullanılan renk uzayına göre değişir. Sadece siyah ve beyazın tonlamalarından oluşan Gray renk uzayı kullanılırsa sadece tek bir katman yeterlidir. Gözelerde bulunan sayı 0-255 arasındadır. Yani 8 bitle ifade edilen bir tamsayıdır. Bu tamsayı değeri o rengin baskınlık değerini verir. Gray renk uzayında 0'dan 255'e doğru gidildikçe beyaz renginin baskınlığı artar, 255de tamamen beyaz olur. Diğer iki renk uzayı ise

bilgisayar dünyasında kullanılan RGB ve RGBA renk uzaylarıdır. RGB ingilizcedeki red(kırmızı), green(yeşil), blue(mavi) renklerinin baş harflerinden gelmektedir. Bu renk uzayında 3 katman olduğu için bu renk uzayını kullanan bir görüntü bilgisayar için 3 katmanlı bir matristir. Matrisin satır ve sütun sayısını ise resmin boyutu verir. Örnek olarak 1280piksele 720 piksel uzunluğunda(1280x720) bir resim 1280 satır, 720 sütundan oluşur. Bu resmi ifade eden matrisin her bir katmanında da 1280 çarpı 720 tane 8 bitlik bir tamsayı değeri bulunur. Diğer renk uzayı olan RGBA'nın RGB'den tek farkı ekstradan bir tane katman içermesidir. Bu katmanda resmin o noktası için saydamlık değeri belirtilir. 0 için tamamen saydam(görünmez), 255 için tamamen mattır. Bu saydıklarım dışında bulunan bazı renk uzayları: HSV, CMYK, YIQ'dur.

2.1.1. OpenCV

OpenCV açık kaynak kodlu bir görüntü işleme kütüphanesidir. İlk olarak C dili kullanılarak geliştirilmiş daha sonra geliştirilmeye C++ dili ile devam edilmiştir. Java, C++, Python dilleriyle kullanılabilir. OpenCV kütüphanesinde ayrıca derin sinir ağı modülü bulunmaktadır. Bu sayede istenildiği takdirde başka bir yapay zeka kütüphanesi kullanmadan sadece OpenCV kullanılarak hem görüntü işleme hem bilgisayarlı görü işlemlerini gerçekleştirebiliriz. OpenCV bulunan diğer başlıca modüller şunlardır:

- **Core:** OpenCV'nin temel fonksiyonlarını ve size, matris, point gibi veri yapılarını bulundurur.
- **HighGui:** Resim görüntüleme, pencereleri yönetme ve grafiksel kullanıcı arabirimleri için gerekli olabilecek metotları barındırır. Bu modül ile örnek tek bir fonksiyon ile bir resmi görüntüleyebiliriz.
- **Imgproc:** Filtreleme operatörleri, kenar bulma, nesne belirleme, renk uzayı yönetimi, renk yönetimi ve eşikleme gibi neredeyse tüm fonksiyonları içine alan bir pakettir.
- **Imgcodecs:** Dosya sistemi üzerinden resim ve video okuma/yazma işlemlerini yerine getiren metotları barındırmaktadır.

OpenCV kütüphanesinde 2500den fazla hazır fonksiyon ve bir çok veri yapısı bulunmaktadır. Ben bunlardan sadece projemde kullandıklarımı açıklayacağım.

Mat veri yapısı: Bilgisayar için resim matrislerden ibaret olduğunu söylemiştim. Mat veri yapısı da bu matrisleri yani resimleri tutan bir yapı olarak düşünülebilir. Bu veri yapısına farklı renk uzaylarında yani farklı katman sayılarında resimleri bellekten okuyarak atayabiliriz. Herbiri için farklı bir tanımlama yapmamıza gerek yoktur. Sıfırdan bir matris tanımlayıp içini kendimiz doldurmak istiyorsak renk uzayı, boyut gibi tanımlamaları kendimiz yapmamız gereklidir.

rectangle fonksiyonu: Imgproc modülünde bulunuyor. Bu fonksiyon ile bir resim üzerinde istenilen yerlere dikdörtgenler çizdirebiliriz. En genel haliyle tanımı şu şekildedir:
void rectangle(Mat mat, Point nokta1, Point nokta2, Scalar renk, int kalınlık)
mat parametresi üzerinde işlem yapılacak resmi ifade eder. nokta1 çizeceğimiz dikdörtgenin başlangıç sınırlarını ifade eden Point türünden bir değerdir. Point sınıfının yapıcı fonksiyonu x ve y koordinatlarını alır. nokta1'e çizeceğimiz dikdörtgenin sırasıyla x ve y başlangıç konumlarını veriyoruz. nokta2'ye ise bitim noktalarını veriyoruz. renk parametremiz Scalar türünden bir değer. Scalar kullanılan renk uzayındaki katman sayısına göre farklı sayıda parametre alıyor. Benim kullandığım renk uzayı 4 katmanlı yani RGBA olduğu için kullandığım yapıcı fonksiyon da 4 parametreliydi. OpenCV bu renk uzaylarını bilindik sıralamada kullanmıyor. RGB uzayından bir renk tanımlarken BGR sırasıyla, RGBA renk uzayından bir renk tanımlarken ABGR sırasıyla tanımlamalıyız. Son parametremiz olan kalınlık da int türünden bir değer. Aldığı değerle doğru orantılı olarak çizdiğimiz dikdörtgenin çizgi kalınlığının artmasını sağlıyor.

putText fonksiyonu: Rectangle fonksiyonu gibi Imgproc modülünde yer alıyor ve onunkine benzer bir işlevi var. Bir resmin üzerine istenilen konumda yazı yazdırılmasına yarıyor.
void putText(Mat resim, Point baslangic, int yazıSitili, int yazıBoyutu, Scalar renk)
İlk parametreye üstünde işlem yapılacak resmi, ikinci parametreye resmin hangi noktasından itibaren yazının yazdırılacağını, 3. parametreye yazıstilini veriyoruz.

Yazı stilleri ImgProc sınıfında static üye değişken olarak tanımlı. Buraya direkt olarak tamsayı değeri yazmamıza gerek yok. 4. parametremiz tamsayı değeri alıyor, yazının boyutunu belirlememize yarıyor. 5. ve son parametremiz ile yazının rengini belirliyoruz.

matToBitmap fonksiyonu: Utils sınıfına ait statik bir fonksiyon. Fonksiyon tanımı şu şekilde: void matToBitmap(Mat resim, Bitmap bitmap)
Fonksiyonu Mat türünden Bitmap türünde dönüştürme yapmak için kullanıyoruz. İlk parametreye mat değerini yani resmimizi veriyoruz. İkinci parametre olarak ise Bitmap türünden boş bir değer vermemiz gerekli. Boş derken null değil. İlk parametrede verdiğimiz Mat nesnesi ile aynı genişlik ve uzunlukta ve aynı renk uzayında olmalı. Benim projemde bu fonksiyonu kullanmamın sebebi OpenCV ile kameradan aldığım resim üzerinde işlemler yapmak için öncelikle resmi Mat türünden ifade etmem gerektiğiydi. Android kısmında ise bir resmi Image View dediğimiz resim göstericilerde göstermek için Bitmap veri yapısını kullanıyoruz. Bu yüzden bu iki tür arasında dönüşüm yapmam için bu fonksiyonu kullanmam gerekti.

Projemde ayrıca Android telefonun kamerasından görüntü alırken OpenCV'nin arayüzlerini ve sınıflarını kullandım.

CvCameraViewListener2 arayüzü: Bu arayüzde 3 tane fonksiyon tanımı bulunur.
Mat onCameraFrame(CameraBridgeViewBase.CvCameraViewFrame inputFrame):
Bu fonksiyon kameradan her görüntü alındığında çalışıyor. Yani kameramız 25fps ile çalışıyor saniyede 25 kere bu fonksiyon çağrılır. Parametresi o an yakalanan görüntüyü verir.
void onCameraViewStarted(int width, int height):
Kamera görüntü almaya başladığı zamanlarda çalışır. Aldığı görüntünün genişlik ve uzunluk değerleniri verir.
void onCameraViewStopped():
Kamera kullanıcının isteğiyle veya herhangi bir sebepten dolayı çalışmayı durdurursa bu fonksiyon çalışır.

JavaCameraView sınıfı: OpenCV ve Android kamera sınıfları arasında köprü görevi görür. Bu sınıfta Camera sınıfından oluşturulmuş mCamera isimli, erişim belirteci protected olan bir üye değişken vardır. Bu değişken ile kameramızın kontrolünü sağlarız. Örnek olarak kameranın görüntüleme boyutunu, yakaladığı resimlerin boyutunu ayarlayabilir, kamera flaşını açıp kapatabilir, kamera efektlerini kullanabiliriz. Zaten Camera sınıfı da Android'in temel kamera sınıfıdır. Dolayısıyla bu üye değişkenle kameranın kontrolü Android işletim sisteminin verdiği derecede bizim elimizde olmuş olur.

Camera.PictureCallback arayüzü: Android Camera sınıfının içinde bulunan statik bir arayüzdür. İçerisinde aşağıdaki fonksiyon tanımını bulundur.

abstract void onPictureTaken(byte[] data, Camera camera)

Bu fonksiyon başaralı bir şekilde kameradan resim alındığı çağrılır. Resim verisi bayt dizisi halinde gelir.

BÖLÜM 3. KONVOLÜSYONEL YAPAY SİNİR AĞLARI İLE BİLGİSAYAR GÖRMESİNİN UYGULANMASI

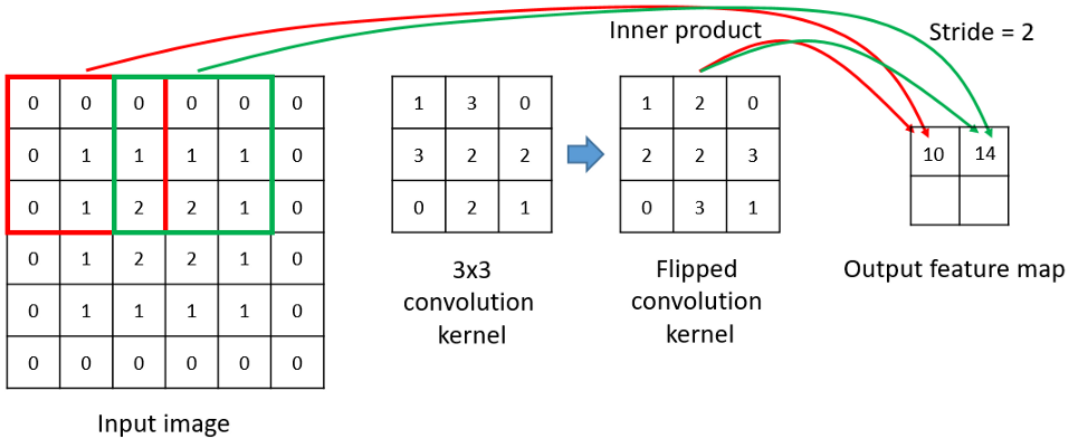
Şekil 3.1. Biz nasıl görürüz – bilgisayar nasıl görür

Klasik yapay sinir ağlarında görüntüden bir bilgi çıkarılmak istendiğinde resim tek katmana indirgenerek tek boyutlu bir matris haline(vektör) getirilerek yapay sinir ağının nöronlarına direkt olarak verilir. Bunun sonucu resmin 2 boyutlu yapısı bozulmuş olur, renk değerleri de siyah beyaz olarak alınacağı için büyük bir veri kaybı söz konusudur. Yapay sinir ağımız bu şekilde istenilen başarıyı elde edemez. Ayrıca hesaplanması gereken ağırlık değerleri kullanılan ara katman sayısı arttıkça büyük oranda artacak ve eğitim süresi uzayacaktır. Bu probleme Yann LeCun isimli mühendis çözüm bulmuştur. Yann LeCun konvolüsyonel katmanları tanımlamıştır.[5]

3.1. Konvolüsyon İşlemi

Konvolüsyon işlemini, konvolüsyonel katmanlarda gerçekleştiririz. Daha önce anlattığımız üzere bilgisayar resimi Şekil 3.2'de gösterildiği gibi matris formunda tutuyordu. Kullanılan renk uzayına bağlı olarak katman sayısı kadar da matrisimiz vardı. Klasik yapay sinir ağı modellerinde bu verileri tek boyutlu bir vektöre indirgediğimizde ciddi veri kayıpları oluşuyordu. Konvolüsyon işlemini yaparken resmi olduğu biçimde alırız. Herbir katmanda belirlediğimiz boyutta ve sayıda olan

filtreler ile konvolüsyon işlemini uygularız. Konvolüsyon işlemine giriş matrisimizin ilk katmanının sağ üst köşesinden başlanır. Belirlediğimiz filtrenin elemanları ile giriş matrisimizin filtre işlemine tabi tuttuğumuz elemanları tek tek çarpılır. Çıkan sonuç çıktı matrisine yazılır.



Şekil 3.2. Resimlere uyguladığımız konvolüsyon işlemi

Daha sonra filtre belirlenen adım sayısı yani Stride kadar sağ kaydırılır. Şekil 3.2'de stride sayısı 2 olarak belirlendiği için filtre 2 adım sağ kaydırılmıştır. Bir sonraki adımda da sağa kayması gerekecektir fakat giriş matrisinin boyutları bunun için yetersizdir. O satır için daha fazla filtreleme işlemi yapılamayacağı görüldüğünde belirlenen stride sayısı kadar satır atlanır ve bir sonraki adıma geçilerek filtreleme işlemine buradan devam edilir. Şekil 3.2'de stride sayısı 2 olduğu için 2 satır atlanacak. Giriş matrisinin uzunluğu, filtre matrisinin uzunluğu ve stride sayısı parametrelerine bağlı olarak çıkış matrisinin hesaplanmasına yarayan formül aşağıdadır:

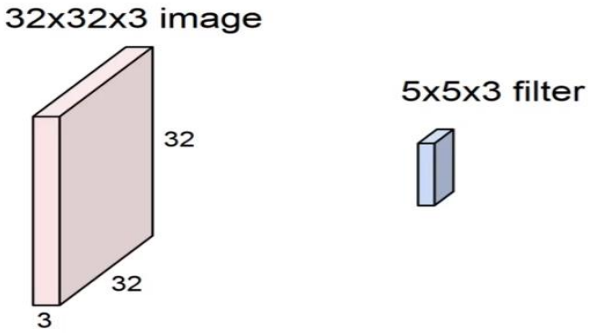
$$U = (N - F) / \text{Stride} + 1 \qquad \textbf{(3.1)}$$

U burada çıkış matrisinin boyutu, N giriş matrisinin boyutu, F uygulanacak filtre matrisinin uzunluğudur. Şekil 3.2'deki örnekte padding ekleme işlemi yapılmıştır. Farklı padding ekleme yöntemleri vardır. Bu örnekte giriş matrisinin tümünü çevreleyecek şekilde 0 değerleri eklenerek 6x6 boyutunda bir matris elde edilmiştir. Asıl giriş matrisi göbekte bulunan sıfırlardan farklı 4x4 boyutunda bir matristir. Bu padding yöntemi için eklenecek padding uzunluğunu (F-1)/2 formülü ile buluruz.

Gerçekten sağlamasını yaparsak (3-1)/2 den 1 değerini bulacağız. Şekil 3.2'de de 1 birim uzunluğunda bir çevreleme yapılmıştır. Padding işlemi ne zaman gerekli ondan bahsedeyim. Bu örnek için giriş matrisimizin orjinali 4x4 boyutunda, filtremiz de 3x3 boyutunda, adım sayımız ise 2'dir. Biz 4x4'lük giriş matrisimizin sağ üst köşesinden başlayarak 3x3 filtre matrisimiz ilk konvolüsyon işlemini yaptıktan sonra stride sayımız 2 olduğu için 2 adım sağa kaymamız gerekecek. Fakat matrisi 2 adım sağa kaydırırsak filtre matrisimizin son sütunu dışarıda kalacak yani boyut uyuşmazlığı olacak. Bu yüzden bu aşamayı es geçip stride sayısı kadar satır atlayıp yine en sağ köşeden konvolüsyon işlemine devam etmemiz gerekecek. Aynı sorunu da burada satırlar için yaşayacağız. Bu sorunun oluşup oluşmayacağını formül1'e gerekli değerleri koyarak da anlayabiliriz. Bizim Şekil 3.2'deki değerlerimiz için sonucumuz tamsayı çıkmayacaktır, bu bize bir sorun olduğunun habercisidir. Ekleyeceğimiz paddingin uzunluğunu (F-1)/2 formülü ile tanımlamıştık. Daha önce verdiğimiz formül herhangi bir padding işlemi gerçekleşmemiş ve gerek yoksa kullanılabilir. Daha genel olarak çıkış matrisinin uzunluğunun hesaplanmasını sağlayan formül aşağıdadır:

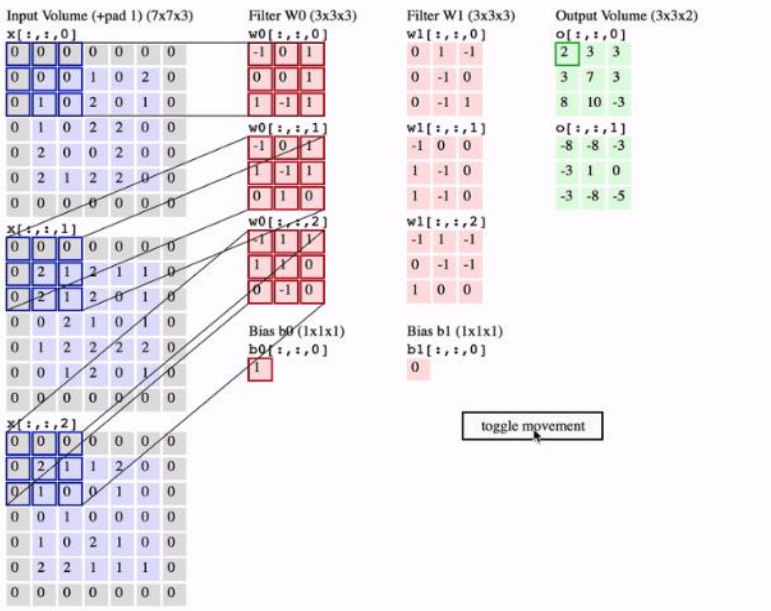
$$C = 1 + \frac{(N+2P-F)}{stride} \qquad P = \frac{(S).(N-1)+(F-N)}{2} \qquad \textbf{(3.2)}$$

Konvolüsyon işleminde girdi olarak alınan resmin ve uygulunacak filtreni katman sayısı aynı olmalıdır. Örnek olarak RGB renk uzayında olan bir resim girdimiz olsun. 3 katmanlı bir matris olacaktır. Uygulayacağımız filtrenin de katman sayısı 3 olmalıdır. Resmin her katmanına filtrenin eşleşen katmanı ile konvulüsyon işlemi yapılır.



Şekil 3.3. Resim ve filtre matrisi örneği

Şekil 3.3'de 32x32 boyutunda 3 katmanlı bir resmimiz vardır. Uygulunan filtre de 5x5 boyutunda ve 3 katmanlıdır. Katman sayılarının aynı olmasının şart olduğundan ve uygulanacak filtrenin boyutunu bizim belirlediğimizi söylemiştim. Buradaki filtre 3x3 boyutunda da olabilirdi. Uygulanacak filtre sayısını da biz belirleriz. 32x32x3'lük giriş matrisimiz, 5x5x3'lik filtremiz olsun, uygulayacağımız filtre sayısı 6, stride sayımız da 1 olsun. Tüm filtreleme işlemleri yapıldıktan sonra çıkış matrisimiz 28x28x6 boyutunda olacaktır. Buradaki 28 değeri formül 1 den hesaplanabilir. Çıkış matrisinin katman sayısının 6 olmasının nedeni ise uyguladığımız filtre sayısının 6 olmasıdır. Yani kaç tane filtre uygulursak çıkış matrisimizin katman sayısı da o olur.



Şekil 3.4. 3 katmanlı bir resme filtre uygulanması

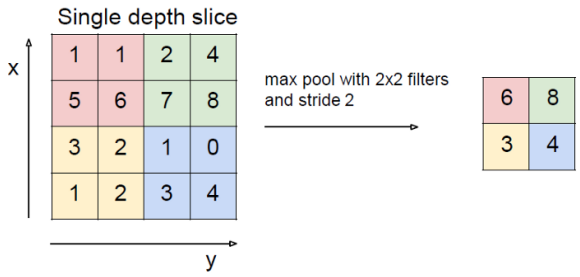
Şekil 3.4 üzerinden yapılan konvolüsyon işlemimi daha detaylıca anlatayım. Elimizde 3 katmanlı bir resim var, her katmana ait değerler 1 birim padding eklenerek alt alta sıralanmış.Onun hemen yanındaki 2 sütunda uygulayacağımız 3 katmanlı filtreler ve bias değerleri var. Son olarak da en sağda çıkış matrislerimiz var. İlk filtremizi resmimize eşleşen katmanlara göre uyguluyoruz. Giriş matrisin en sağından başlayarak konvolüsyon işlemi yapıyoruz. Her adım için her katmanda konvolüsyon işlemi sonucu bulduğumuz değerleri topluyoruz ve buna bias değeri ekliyoruz. Çıkan sonuç bizim çıktı matrisimizin ilk elemanı oluyor. Bu şekilde tüm resmi tarayarak çıktı matrisini elde ediyoruz. Aynı işlemleri diğer filtre ile yapınca da

diğer çıkış matrisimi elde etmiş oluyoruz. Çıktı matrisimiz 3x3x2 boyutunda oldu. Çıkış matrisimizin boyutu azaldı, eğer bunun olmamasını istiyorsak işlemlerden önce daha fazla padding ekleyebiliriz.

Konvolüsyon işlemi ile resimden bilgi çıkarmak hedeflenir. Çıkarılan bilgiye örnek olarak kenar tespiti verilebilir. Kenar tespiti için oluşturulmuş özel fitreler vardır. Bu filtreler uygulanarak resimde kenar tespiti yapılır. Bu tespit genelde CNN(Convolutional Neural Networks)'nin ilk katmanlarında yapılır.

3.2. Ortaklama(Pooling) İşlemi

Bu işlem pooling katmanlarında yapılır. Bu katmanda öğrenme yoktur. Amacı giriş matrisinin katman sayısını sabit tutarak parametre sayısını yani boyutunu azaltmaktır. Farklı şekillerde yapılabilir. En genel kullanımı maksimum ortaklama yapmaktır. Hesaplama karmaşıklığını azaltır. Ancak Hinton'ın kapsül teorisine göre verideki önemli bazı bilgilerin de kaybolmasına sebep olduğu için başarımdan ödün vermektedir.



Şekil 3.5. Maksimum ortaklama

Şekil 3.5 üzerinden anlatacak olursak, 4x4 boyutunda olan giriş matrisimize 2x2 boyutunda bir boş matris ile pooling uygulanmıştır. Giriş matrisi 2x2 boyutunda küçük matrislere bölünmüştür. Bu 2x2 matrisin elemanları içinde en büyük olan eleman çıkış matrisine yazılır. Bu örnekte boyut yarıya düşmüştür. Pooling katmanları genelde gerekli görüldüğü yerde konvolüsyon katmanından sonra konulur.

Konvolüsyonel sinir ağları günümüzde bilgisayarlı görünün bel kemiğini oluşturmaktadır. CNN'ler ile resim sınıflandırma, resim analizi, nesne tespiti

yapılabilir. Ben sadece projemde kullanmış olduğum nesne tespiti üzerinde duracağım.

3.3. Nesne Tespiti

Bilgisayarlı görüde sıkça yapılan bir işlemdir. Bir resimdeki nesnelerin konumlarıyla birlikte tespit edilmesidir. Konum bilgisi genelde nesnenin dikdörtgen içine alınmış halidir. Nesne tanıma yaparken akla gelen en basit yöntem resimden farklı boyutta parçalar alıp bunları CNN'den geçirerek o parçalarda sinir ağımızın eğitilmiş olduğu nesnelerden olup olmadığının kontrol edilmesidir. Bir resimde farklı boyutlarda, farklı uzaklıklarda nesneler olabilir. Bizim bu kaba kuvvet tekniğiyle hiçbir noktayı kaçırmadan tüm resmi olabilecek tüm boyutlarda parçalara bölüp bunu CNN'e verip sonuç almamız mümkün değildir. Bu inanılmaz bir işlem gücü gerektirir, yapılması mantıklı değildir. Bunun çözümü ise bölge önerisidir. Bölge önerisinin tanımı bir resim üzerinde nesnelerin bulunması olası bögelerin tespit edilmesidir. Bu bölgeler yaklaşık 2000 adettir. Bu bölgeler resimde ton farkının çok fazla olduğu yerler sayesinde tespit edebilir. Selective Search algoritması ile gerçekleştirilir.

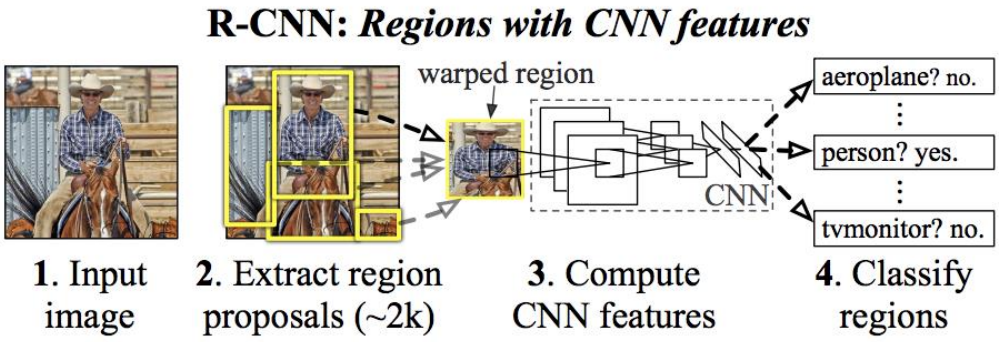
3.3.1. Region-Based CNN's(R-CNN)

Önceki kısımda bahsettiğimiz bölge önerisi algoritmalarını kullanan konvolüsyonel ağlar R-CNN olarak adlandırılır. Klasik R-CNN, Fast R-CNN, Faster R-CNN olmak üzere 3 çeşiti vardır.

3.3.1.1. Klasik R-CNN

Klasik R-CNN'de bölge önerisi ile oluşturulan 2000 kadar resim parçasının tamamı konvolüsyonel sinir ağından geçirilir. Bölge önerisi ile oluşturulan parçalar farklı boyutlarda olabilir. CNN kendisine gelen girdilerin aynı uzunlukta olmasını bekler, bu yüzden bölge önerisi ile bulunan parçalar CNN'e verilmeden önce aynı boyuta getirilir. Konvolüsyon işleminden geçen resim SVM(Support Vector Machine) ile

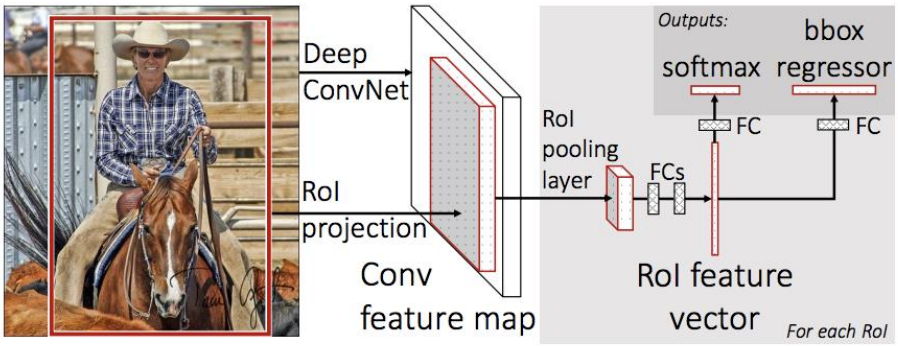
sınıflandırılır yani hangi nesne olduğu tespit edilir. Lineer regresyon ile de nesnenin sınırları tespit edilir. Bu şekilde 2000 kadar resim parçası üzerinde bu saydığım işlemlerin yapılması çok zaman alan bir süreçtir. Gerçek zamanlı nesne tanıma klasik R-CNN'de imkansızdır.[6]



Şekil 3.6. Klasik R-CNN adımları

3.3.1.2. Fast R-CNN

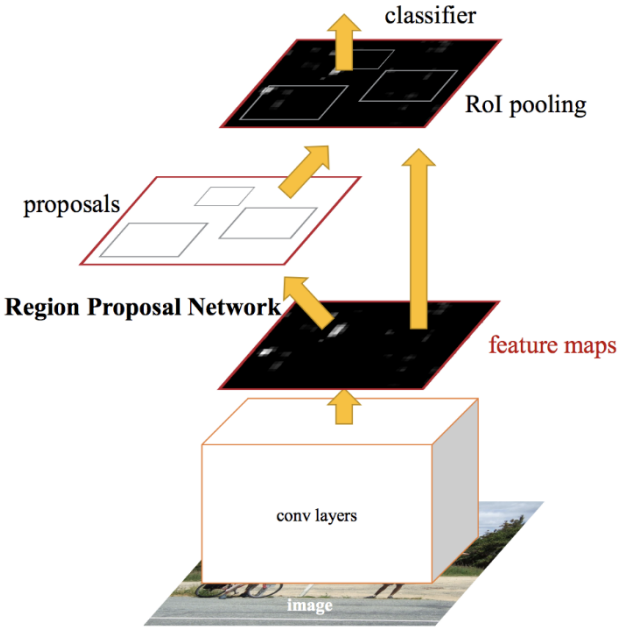
Fast R-CNN'de resim parçalara bölünmeden önce CNN'e verilir. Daha sonra girdi resmimize göre bir özellik haritası çıkarılır. Bölge önerileri bu özellik haritası üzerinden yapılır. Çıkarılan bölgelerin boyutları eşitlenerek fully connected layera verilir. Bu katmandan çıkan sonuçlara göre sınıflandırma yapılır ve sınırlar belirlenir. Sınıflandırma için softmax, sınırlar için lineer regresyon kullanılır. Klasik modeldeki gibi her bir parçayı CNNetden gerçirmediği için çok ciddi bir performans artışı söz konusudur.



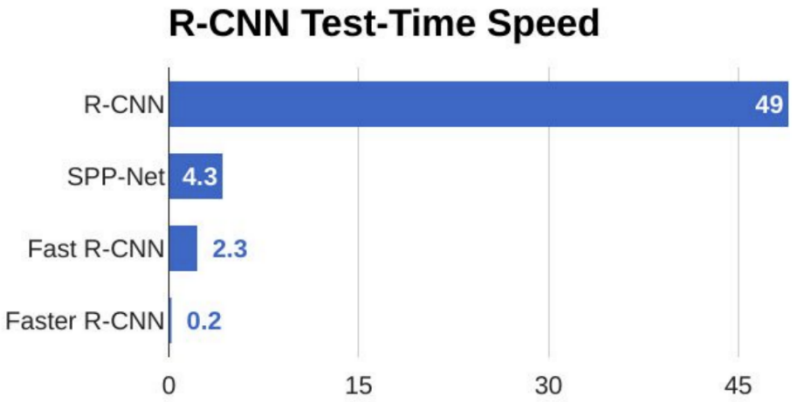
Şekil 3.7. Fast R-CNN adımları

3.3.1.3. Faster R-CNN

Faster R-CNN'de de Fast R-CNN'de olduğu gibi resim ilk olarak CNNet'ten geçirilerek bir özellik haritası çıkarılır. Bu aşamadan sonra bölge önerilerini önceki modellerde olduğu gibi bir algoritma ile değil de farklı bir konvolüsyonel sinir ağında buluruz. Belirlenen bölgeler aynı boyuta getirilerek fully connected layera verilir. Burada sınıflandırma ve lineer regresyon işlemleri yapılır. Faster R-CNN'de diğer modellere ek olarak bölge önerisi için tasarlanan ağın da eğitilmesi gereklidir, Fakat bu işlem bölge önerisini algoritmalar ile yapmaktan daha kısa sürer. Bunun sonucu olarak yine ciddi bir performans artışı söz konusudur. Şekil 3.9'da bu 3 modelin eğitim zamanlarının karşılaştırılması gösterilmiştir.



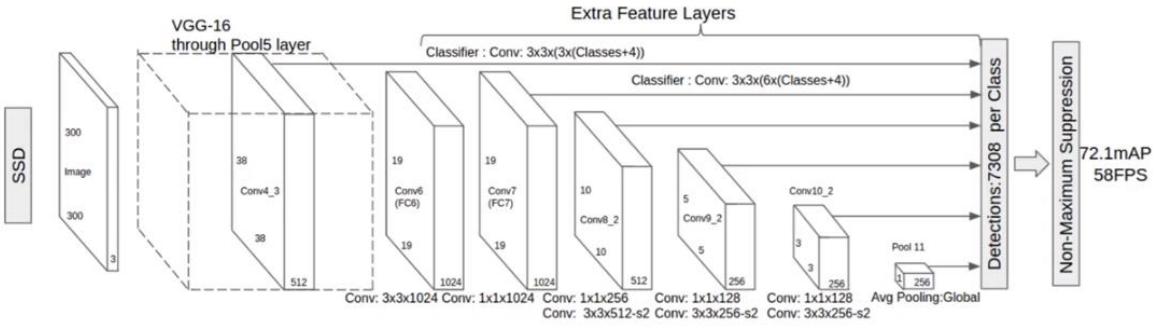
Şekil 3.8. Faster R-CNN adımları



Şekil 3.9. Eğitim zamanı karşılaştırmaları

3.3.1.4. Single shot multibox detecter(SSD)

R-CNN'deki gibi her bölge için ayrı hesaplama yapmak yerine, tek bir seferde tüm resimi tarar. Bu özelliğinden dolayı R-CNN çeşitlerinden daha hızlıdır fakat test doğruluk oranı daha azdır. Ben bir mobil uygulama yaptığım için daha düşük donanımlarda çalışmaya elverişli olan bu modeli seçtim.



Şekil 3.10. SSD Mimarisi

3.3.2. Tensorflow

Açık kaynak kodlu bir deep learning(derin öğrenme) kütüphanesidir. Esnek yapısı sayesinde, tek bir API ile platform farketmeksizin hesaplamaları, bir veya birden fazla CPU, GPU kullanarak deploy etmenize olanak sağlar. Temelinde Python kullanılarak geliştirilen bu framework, günümüzde Python'ın yanısıra C++, Java, C#, Javascript ve R gibi birçok dili desteklemektedir.[7]

Web tarayıcı, kişisel bilgisayarlar, mobil cihazlar/IOT cihazları üzerinde çalışabilmesi farklı varyasyonları vardır.

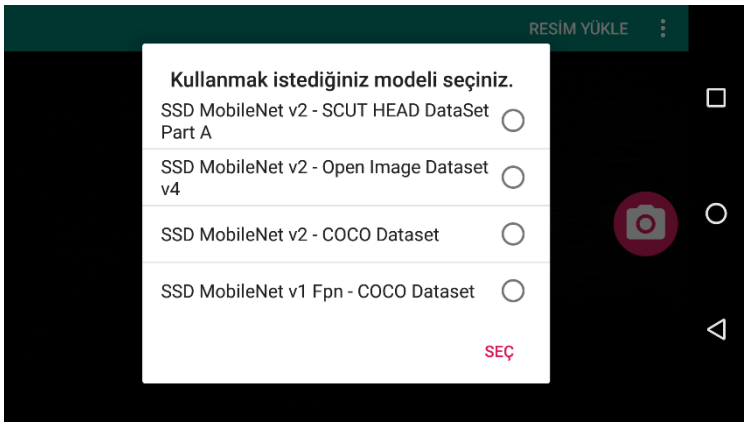
Projemde Tensorflow kütüphanesini kullandım.

BÖLÜM 4. UYGULAMA

Şuana kadar projemde kullandığım teknolojilerden bahsettim. Şimdi de projemde önemli gördüğüm kod kısımlarından bahsedeceğim. Tasarladığım mobil uygulama tek bir activity oluşuyor. Activityler iki kısımdan oluşur. Bir parçası görsel tasarımların gerçekleştirildiği bir xml dosyasıdır, diğer kısımda bu görsel nesnelerin kontrolünün sağlandı Java sınıfıdır. Sınıf üzerinden de dinamik olarak görsel nesne oluşturmak mümkündür. Activity tasaramım genel anlamda bir toolbar ve buna entegre edilmiş bir menüden, bir adet floating action buttondan, bir adet cameraview ve bir adet imageviewden oluşuyor. Cameraview ve imageview birbirleriyle içiçe geçmiş halde bulunuyor. Gerektiği kısımlarda biri aktif edilip diğeri gizleniyor.

4.1. Uygulamanın Çalıştırılması

Uygulamam ilk açıldığında bir alertview ve bunun içine eklenmiş listview karşımıza çıkıyor. Buradan programda nesne tanıma yapılırken kullanmak istediğimiz modeli seçiyoruz. Aslında genel anlamda tüm modeller SSD modeli fakat SDD modelinin farklı varyasyonları olduğu için ve farklı veri setleri ile eğitildikleri için herbiri farklı sonuçlar vermekte.

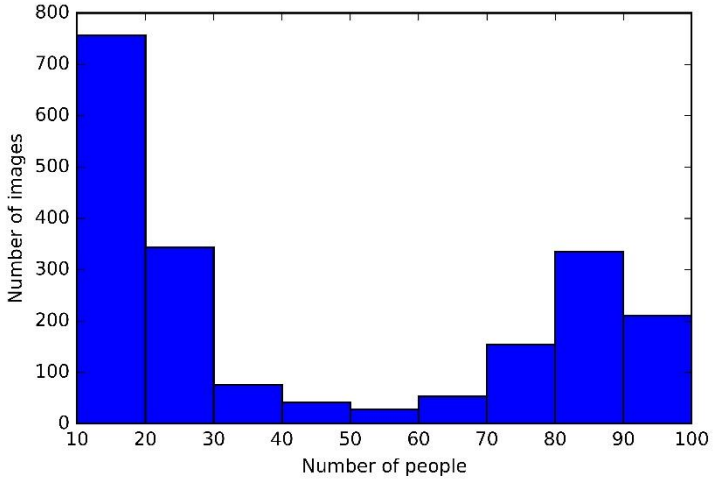


Şekil 4.1. Uygulama açıldığında karşılaşılan ekran

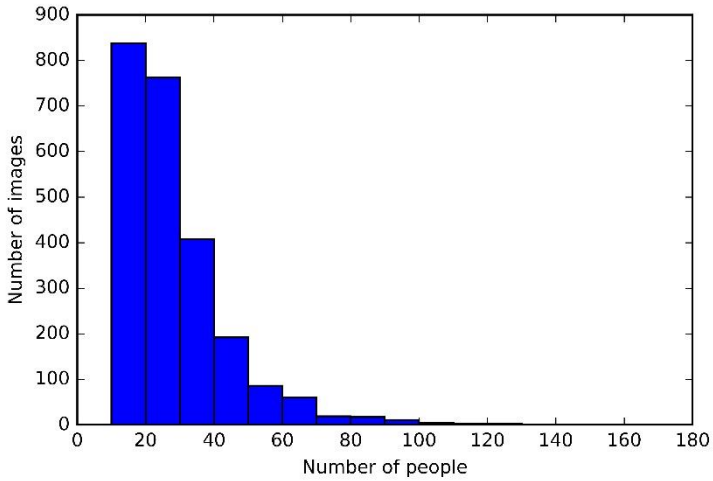
Listeden seçebileceğimiz ilk model SSD MobileNet v2 – SCUT HEAD DataSet Part A. Bu model üstünde eğitimi önceki bölümde bahsettiğim Tensorflow kütüphanesi

ile GPU kullanarak ben gerçekleştirdim. Modelimi yaklaşık 42bin epoch kadar eğittim. Resimlerden 1400 tanesini eğitime, 600 tanesini teste ayırdım.

SCUT HEAD veriseti, Part A ve Part B şeklinde 2'ye bölünmüş. PartA'da 2000 kadar resim ve toplam 67321 tane insan kafası var. Bunlar etiketlenmiş bir halde bize sunulmuş. Part B'de ise 2405 resim ve 43940 insan kafası var. Bu iki veri setindeki insan sayısının resim sayısına göre dağılımını Şekil 4.2 ve Şekil 4.3'de görebilirsiniz.



Şekil 4.2. SCUT HEAD Dataset – Part A



Şekil 4.3. SCUT HEAD Dataset – Part B

Listedeki 2.model olan SSD MobileNet v2 – Open Image Dataset v4, Open Image datesetinin küçük bir parçası ile eğitilmiş hazır bir model. İçerisinde 600 tane sınıf bulunduruyor ve eğitim 1,743,042 tane sınırları çizilmiş nesne ile gerçekleştirilmiş.

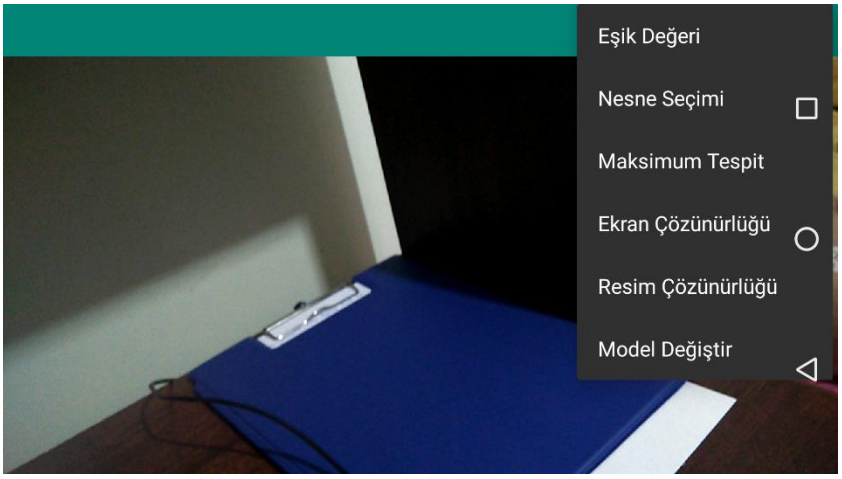
3 ve 4. Modeller COCO Dataseti üzerinde eğitilmiş. Bu modeller 91 tane farklı sınıfı tanıyabiliyor. 4. Model, 3.farklıdan olarak FPN mimarisi ile tasarlanmış ve girdi olarak aldığı resim 640x640 boyutunda. Diğer modellere göre daha iyi sonuç veriyor ama daha yavaş. Diğer modeller 300x300 boyutunda resimleri kabul ediyordu. Tüm modellerde resmi ağımıza vermeden önce yeniden boyutlandırma işlemi yapıyoruz. Kullandığım modellerin ve daha fazlasının test anındaki hız ve başarılarının listelendiği tablo aşağıdadır. Herbir model COCO veriseti ile eğitilmiştir.

Tablo 4.1. Nesne tespit modellerinin karşılaştırılması

Model	Speed (ms)	Başarı Oranı
SSD MobileNet v1	30	21
SSD MobileNet v1 FPN	56	32
SSD MobileNet v2	31	22
SSD Inception v2	42	24
Faster R-CNN Inception v2	58	28
Faster R-CNN Resnet101	106	32
Faster R-CNN NAS	1833	43

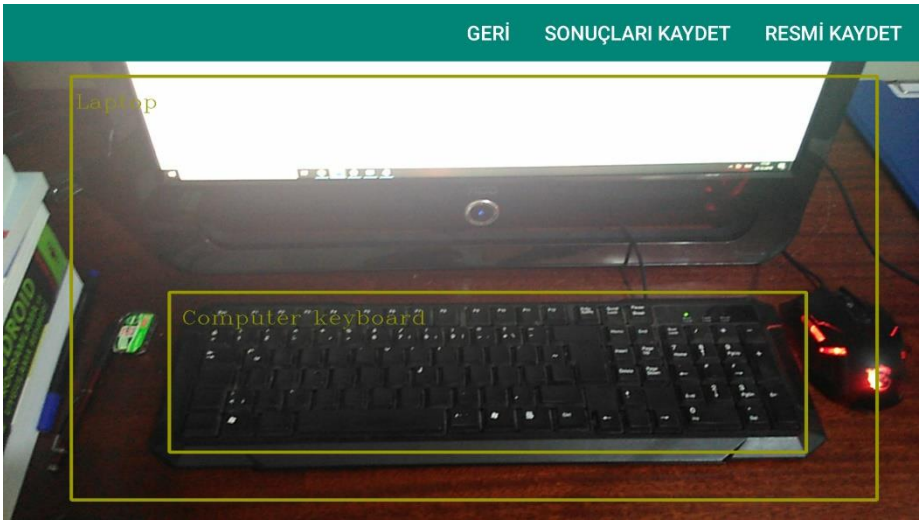
Model seçimini yaptıktan sonra modelimiz yükleniyor ve kamera, resim yakalamak için bir button ve ekranın üstünde bulunan bir toolbar bizi karşılıyor. Toolbarda bulunan menümüzde 7 farklı menü elemanı var. “RESİM YÜKLE” ye tıklayarak telefonumuzda daha önce kayıtlı olan bir resim üzerinde nesne tespiti yapabiliriz. Eşik Değeri kısmından eşik değerini değiştirebiliriz. Nesne Seçimi kısmından modelimizin tespit edebildiği nesneleri görebiliriz ve istersek bunlardan sadece seçtiklerimizi bulmak istiyorsak ilgili nesneleri seçerek bunu sağlayabiliriz. Maksimum Tespit ile resimde tespit edilecek maksimum nesneyi belirleyebiliriz. Ekran çözünürlüğünden kameramızın ekrana verdiği görüntünün çözünürlüğünü

ayarlayabiliriz. Resim çözünürlüğünden yakaladığımız resmin hangi çözünürlükte olacağını belirleyebiliriz. Model değiştir kısmından ise modelimizi değiştirebiliriz.



Şekil 4.4. Uygulama ana ekranı ve menüsü

Bu aşamada bir fotoğraf yükler veya yakalarsak bu fotoğraf direkt olarak ağımıza veriliyor, sonuçlar alınıyor, tespit edilen nesneler resim üzerinde kare içine alınıyor ve adları yazılıyor. İstersek bu çıktı resmini Resmi Kaydet'e tıklayarak kaydedebiliriz. Ayrıca elde edilen sonuçları Sonuçları Kaydet'e tıklayarak bir metin belgesi şeklinde kaydedebiliriz.



Şekil 4.5. Uygulama sonuç ekranı ve menüsü

4.2. Önemli kod kısımlarının açıklanması

Bu kısımda önemli gördüğüm kod kısımlarını seçip bu kısımlardaki önemli gördüğüm kodları numarandıracak ve açıklayacağım.

4.2.1. resimIslemleri fonksiyonu

```
public void resimIslemleri(Bitmap image)
{
1   Bitmap boyutlandirilmis = Bitmap.createScaledBitmap(image,
            aktivite.getResizeGenislik(),aktivite.getResizeUzunluk(), filter: false);
    long start,end;
    start = System.currentTimeMillis();
2   tumSonuclar = MainActivity.getClassifier().analizEt(boyutlandirilmis,
            aktivite.getSecilenSiniflar(),aktivite.getSeekBar().getProgress(),
            esik: aktivite.getSeekBarEsik().getProgress()/100.0f);
    end = System.currentTimeMillis();
    Log.e( tag: "test zamani", msg: ""+(end-start));

3   List<Classifier.TumSonuclar.TespitEdilenler> liste = tumSonuclar.getListe();

    Log.e( tag: "sonuclar", tumSonuclar.toString());

    int uzunluk = image.getHeight();
    int genislik = image.getWidth();
4   Mat mat = new Mat(uzunluk,genislik, CvType.CV_8UC4); //4

    Utils.bitmapToMat(image,mat);//5

    Log.e( tag: "getSiniriGecenSayisi",String.valueOf(tumSonuclar.getSiniriGecenSayisi()));
5   for(int i = 0; i< tumSonuclar.getSiniriGecenSayisi(); i++)
    {
        RectF kare = liste.get(i).getKutu();
        Scalar renk = renkler[random.nextInt(renkler.length)];
        String yazi = liste.get(i).getSinif();
        Imgproc.rectangle(mat,new Point( x: kare.left*genislik, y: kare.top*uzunluk),
                new Point( x: kare.right*genislik, y: kare.bottom*uzunluk),renk, thickness: 3);
        Imgproc.putText(mat,yazi,new Point( x: kare.left*genislik+yazi.length()*fontScale,
                y: kare.top*uzunluk+(fontScale*50)),Imgproc.FONT_HERSHEY_COMPLEX,fontScale,renk);
    }

6   ciktiResim = Bitmap.createBitmap(genislik,uzunluk, Bitmap.Config.ARGB_8888);

    Utils.matToBitmap(mat,ciktiResim);

    aktivite.getImageView().setImageBitmap(ciktiResim);

    this.setVisibility(GONE);
    this.setEnabled(false);

    aktivite.popupGecis(false,true);
    aktivite.resmiAktifEt();
}
```

Şekil 4.6. resimIslemleri fonksiyonu

1 numaralı kısımda parametre olarak aldığım resmi ağıma vermeden önce yeniden boyutlandırıyorum.

2 numaralı kısımda yeniden boyutlandırdığım resmi ağıma veriyorum. Bu işlemi analizEt fonksiyonu ile gerçekleştiriyorum. Bu fonksiyona sırasıyla boyutlandırılmış resmimi, tespit etmek istediğim sınıfları, maksimum tespit edilmesini istediğim nesne sayısını ve eşik değerini parametre olarak veriyorum. Bu fonksiyon bana TumSonuclar türünden bir nesne döndürüyor. Bunu "tumSonuclar" isimli TumSonuclar turunden referansım ile yakalıyorum.

3 numaralı kısımda tumSınıflar referansım üzerinden tespit ettiğim nesnelerin listesine ulaşıyorum.

4. numaralı kısımda OpenCV kütüphanesi ile bazı işlemleri gerçekleştirebilmek için Bitmap türünde olan resmimi Mat türüne çeviyorum. Bu kısımların ayrıntılarından daha önceki bölümlerde bahsetmiştim.

5 numaralı kısımda eşik değerini geçen nesne tespiti sayısı kadar çalışan bir döngüm var. Bu döngü ile eşik değerini geçen her nesnenin sınırlarını orjinal resmim üzerine çizdiriyor ve sınıf isimlerini yazdırıyorum.

6 numaralı kısımda üzerinde işlem yaptığım orjinal resmimi tekrar Bitmap türüne getiriyor, imageviewime bu resmi veriyorum. Ayrıca kamerayı görünmez ve pasif hale getiriyor, toolbardaki menümdeki bazı kısımları görünmez bazı kısımları görünür hale popupGecis fonksiyonu sayesinde getiriyorum. Son olarak sonuç resmini gösterdiğim imageviewi aktif hala getiriyorum.

4.2.2. Create fonksiyonu

```
public static Classifier create(final AssetManager assetManager,
                                 final String modelFilename, final String labelFilename,
                                 final int inputSizeX,final int inputSizeY) throws IOException
{
  1
  final TensorFlowObjectDetectionAPIModel d = new TensorFlowObjectDetectionAPIModel();
  2
  InputStream labelsInput = null;
  labelsInput = assetManager.open(labelFilename);
  BufferedReader br = null;
  br = new BufferedReader(new InputStreamReader(labelsInput));
  String line;
  while ((line = br.readLine()) != null)
  {
    d.labels.add(line);
  }
  br.close();

  3d.inferenceInterface = new TensorFlowInferenceInterface(assetManager, modelFilename);
```

Şekil 4.7. create fonksiyonu

```
4   d.outputNames = new String[] {"detection_boxes", "detection_scores",
                                  "detection_classes", "num_detections"};
    d.intValues = new int[d.inputSizeX * d.inputSizeY];
    d.byteValues = new byte[d.inputSizeX * d.inputSizeY * 3];
    d.outputScores = new float[MAX_RESULTS];
    d.outputLocations = new float[MAX_RESULTS * 4];
    d.outputClasses = new float[MAX_RESULTS];
    d.outputNumDetections = new float[1];
    return d;
}
```

Şekil 4.8. create fonksiyonu devamı

Şekil 4.7 ve Şekil 4.8'deki fonksiyon Tensorflow kütüphanesinin resmi örnek uygulamalarından alınmıştır. Üzerinde çok fazla değişiklik yapmadım ama genel olarak açıklamaya çalışacağım. Bu fonksiyon ağımız ile aramızdaki iletişimi sağlayan TensorflowObjectDetectionAPIModel isimli sınıfımızdan bir nesne oluşturulmasına yarıyor. Bu fonksiyonu her farklı bir model yüklediğimizde çağırmamız gereklidir. 1 numaralı kısımda belirtilen türde bir nesne oluşturuyoruz. 2 numaralı kısımda modelimizin eğitildiği verisetine göre değişiklik gösteren, içinde modelimizin eğitildiği sınıf türlerini tutan labelmap dosyamızın okunarak labels isimli vektörümüze verildiğini görüyoruz. 3 numaralı kısımda modelimiz yükleniyor. Son olarak 4 numaralı kısımda ise çıktı isimlerimizi tanımlıyor, çıktılarımızı tutacak dizileri oluşturuyoruz. Çıktılarımızı tutacak dizileri MAX_RESULTS sabiti ile oluşturuyoruz. Bu sabitin değerinden fazla çıktıyı bize ağımız veremez.

4.2.3. analizEt Fonksiyonu

Bu fonksiyonda kameradan yakaladığımız veya dosya sisteminden yüklediğimiz resmimizi ağımıza veriyoruz.

1 numaralı kısımda Bitmap türündeki resmimizi intValues isim integer türünden dizimize veriyoruz. Yani burada bir tür dönüşümü var.

2 numaralı kısımda da bir dönüşüm gerçekleştiriliyor. Integer türü 32 bit uzunluğunda, byte türü ise 8 bit. Biz önceki adımda resmimizin 3 katmanının değerlerini(RGB) bitmap türünden integer dizisine dönüştürmüştük. Dizideki her elemanın düşük anlamlı ilk 24 biti bu 3 katmanın değerlerini tutuyor. Biz de bu döngü içerisinde maskeleme ve sağa kaydırma işlemleri ile bu değerleri alıp bir byte

dizisine veriyoruz. Buradan modelimizin byte türünden değerleri girdi olarak kabul ettiğini anlayabiliriz.

3 numaralı kısımda modelimize veri beslemesini feed fonksiyonu ile gerçekleştiriyoruz. Bu fonksiyona girdi olarak alınacak resmin yeniden boyutlandırılmış halinin boyutunu ve katman sayısını veriyoruz.

4 numaralı kısımda önceki adımda beslediğimiz verileri ağımızdan geçirelerek sonuçları elde ediyoruz.

5 numaralı kısımda her yeni bir resim analiz edildiğinde çıktıları tutacak dizilerin tekrar oluşturulmasını istediğimiz için dizilerimizi oluşturuyoruz.

6 numaralı kısımda bulduğumuz sonuçları yakalayıp oluşturduğumuz dizilere atıyoruz. Fetch fonksiyonun ilk parametresine yakalamak istediğimiz çıktı adını, 2. parametresine bu çıktıların hangi diziye aktarılacağını veriyoruz.

7 numaralı kısımda önceki kısımda dizilere aktardığımız tamamı float türünden olan çıktıları, oluşturduğum sonuçları düzenli ve anlamlı bir şekilde kullanmamıza, görebilmemize yarayan TumSonuclar adlı sınıfın yapıcı metoduna veriyoruz. Son olarak bu yapıcı metottan dönen nesneyi return ediyoruz.

```
public TumSonuclar analizEt(final Bitmap bitmap, ArrayList<String> secilenler,
                            int maxTespit, float esikDegeri)
{

  Trace.beginSection( sectionName: "recognizeImage");
  Trace.beginSection( sectionName: "preprocessBitmap");

1 bitmap.getPixels(intValues,  offset: 0, bitmap.getWidth(),
          x: 0,  y: 0, bitmap.getWidth(), bitmap.getHeight());

2 for (int i = 0; i < intValues.length; ++i)
  {
    byteValues[i * 3 + 2] = (byte) (intValues[i] & 0xFF);
    byteValues[i * 3 + 1] = (byte) ((intValues[i] >> 8) & 0xFF);
    byteValues[i * 3 + 0] = (byte) ((intValues[i] >> 16) & 0xFF);
  }
  Trace.endSection();

  Trace.beginSection( sectionName: "feed");
3 inferenceInterface.feed(inputName, byteValues,  ...dims: 1, inputSizeX, inputSizeY, 3);
  Trace.endSection();

  Trace.beginSection( sectionName: "run");
4 inferenceInterface.run(outputNames, logStats);
  Trace.endSection();

5 Trace.beginSection( sectionName: "fetch");
  outputLocations = new float[MAX_RESULTS * 4];
  outputScores = new float[MAX_RESULTS];
  outputClasses = new float[MAX_RESULTS];
  outputNumDetections = new float[1];
6 inferenceInterface.fetch(outputNames[0], outputLocations);
  inferenceInterface.fetch(outputNames[1], outputScores);
  inferenceInterface.fetch(outputNames[2], outputClasses);
  inferenceInterface.fetch(outputNames[3], outputNumDetections);
  Trace.endSection();

7 return new TumSonuclar( tfapi: this,outputLocations,outputClasses,outputScores,
        outputNumDetections,secilenler,maxTespit,esikDegeri);
}
```

Şekil 4.9. analizEt fonksiyonu

4.2.4. TumSonuclar sınıfı

```
class TumSonuclar {
  private int tespitSayisi;
  private int siniriGecenSayisi;
  private List<TespitEdilenler> liste;
  private float esikDegeri;
  private int id;
  private int sayac;
  private TensorFlowObjectDetectionAPIModel tfapi;
  private int maksTespit;

  public TumSonuclar(TensorFlowObjectDetectionAPIModel tfapi, float[] _konum,
                     float[] _sinif, float[] _skor,
                     float[] _tespitSayisi,ArrayList<String> secilenler, int maksTespit, float esikDegeri) {

    this.id = sayac++;
    this.tfapi = tfapi;
    this.maksTespit = maksTespit;
    this.esikDegeri = esikDegeri;

    liste = new ArrayList<TespitEdilenler>();
    tespitSayisi = (int)_tespitSayisi[0];
1   for(int i=0;i<tespitSayisi && siniriGecenSayisi < maksTespit ;i++)
    {
2     if(_skor[i] < esikDegeri || (secilenler.size() != 0
              && !secilenler.contains(tfapi.labelIndexAt( indeks: (int)_sinif[i]-1))))
        continue;
3     liste.add(new TespitEdilenler(new RectF(_konum[4*i+1],_konum[4*i],
              _konum[4*i+3],_konum[4*i+2]),_skor[i],tfapi.labelIndexAt( indeks: (int)_sinif[i]-1)));
4     siniriGecenSayisi++;
    }
  }
```

Şekil 4.10. TumSonuclar sınıfı

Oluşturduğum bu sınıf ağın bana verdiği çıktıları düzenleyerek kullanması ve okunması daha kolay bir hale getiriyor. Kendi içinde TespitEdilenler isminde bir sınıf daha bulunduruyor. Bu sınıf tespit edilen nesneleri tutmaya yarıyor. TumSonuclar sınıfı daha genel bir sınıf. Üye değişkenleri ile tespit sayısını, sınırı geçen nesne sayısını, sınırı geçen ve tespit edilen tüm nesnelerin listesini tutuyor. Ağımızdan çıkan verileri ve uygulamadan aldığımız kullanıcının seçimine bağlı olan eşik değeri, maksimum tespit gibi değerleri bu sınıfın yapıcı fonksiyonuna vererek tüm sonuçları bir seferde düzenliyoruz.

1 numaralı kısımda döngü şartımızı oluşturuyoruz. Tespit sayısı aşılmadıkça ve sınırı geçen(eşik değerini geçen) nesne sayısı yine kullanıcıdan aldığımız maksimum tespit sayısından küçükse bu döngü çalışacak.

2 numaralı kısımda bir if sorgumuz var. Eğer skor değeri eşik değerinden küçükse veya kullanıcı özel bir sınıfın seçilmesi için uygulamadaki menüden bir seçim yapmışsa o an üzerinde işlem yaptığımız nesne kullanıcının seçtiği nesneler içinde mi diye kontrol ediyoruz. İlk şart sağlanıyor ve/veya ikinci şart sağlanmazsa döngünün başına dönüyoruz.

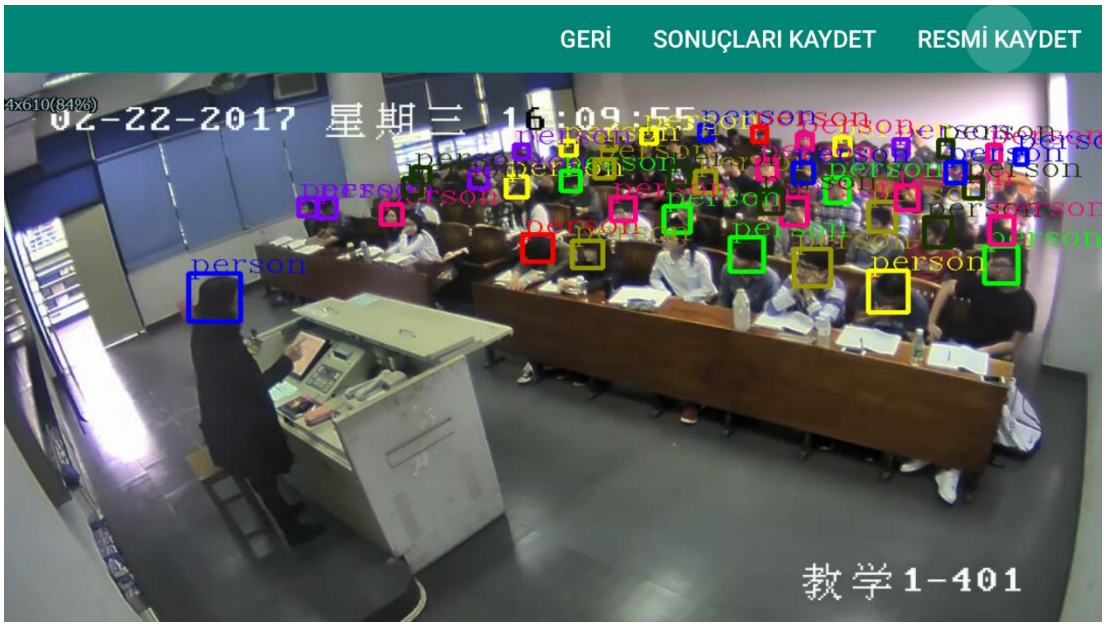
3 numaralı adımda, 2 numaralı if sorgusundan geçmeyi başarabilen nesneleri listemize ekliyoruz.
4 numaralı kısımda sınırı geçen nesne sayısını tutan siniriGecenSayisi değişkenini bir arttırıyoruz.

Aşağıdaki resimde de daha önce bahsettiğim TumSonuclar sınıfının içinde bulunan TespitEdilenler sınıfının tanımı ve TumSonuclar sınıfının tanımının devamı var. Burada önemli gördüğüm bir kısım olmadığı için açıklamayacağım. Basit bir veri yapısı oluşturdum sadece.

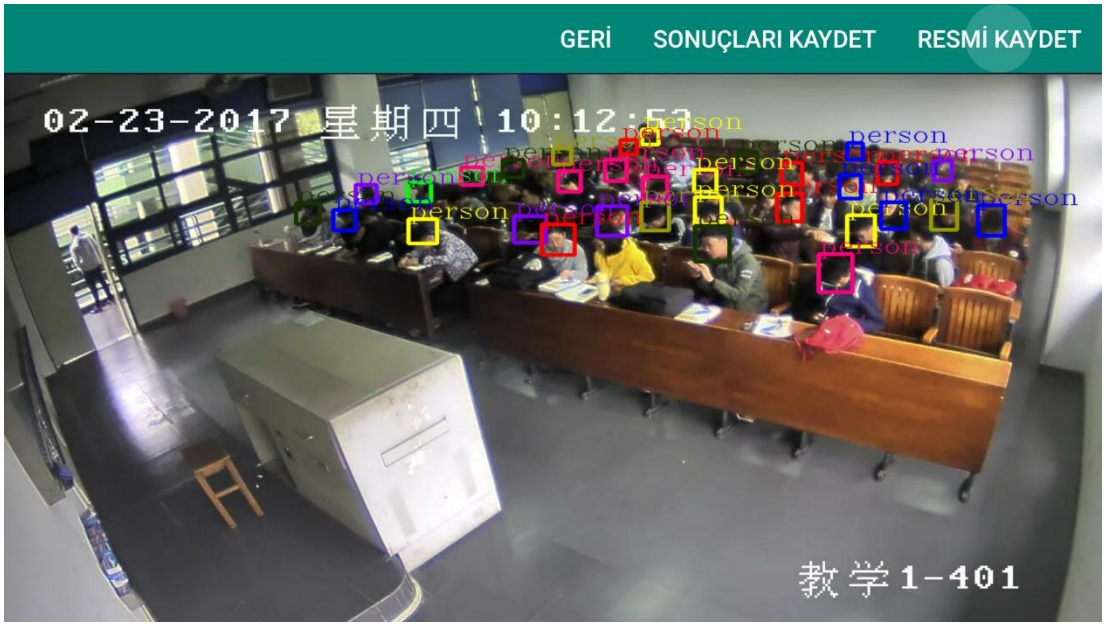
```
class TespitEdilenler
{
  private RectF kutu;
  private float skor;
  private String sinif;

  public TespitEdilenler(RectF kutu, float skor, String sinif) {
    this.kutu = kutu;
    this.skor = skor;
    this.sinif = sinif;
  }

  public RectF getKutu() { return kutu; }

  public float getSkor() { return skor; }

  public String getSinif() { return sinif; }

  @Override
  public String toString() {
    return "Tespitler{" +
            "kutu=" + kutu +
            ", skor=" + skor +
            ", sinif='" + sinif + '\'' +
            '}';
  }
}

public int getTespitSayisi() {
  return tespitSayisi;
}

public int getSiniriGecenSayisi() { return siniriGecenSayisi; }

public List<TespitEdilenler> getListe() { return liste; }

@Override
public String toString() {

  String cikti = "Tespit sayisi = " + tespitSayisi + "\nSiniri gecenlerin sayisi = " + siniriGecenSayisi +
          "\nEsik değeri = " + esikDegeri + "\n<---------------------------->";
  for(int i=0;i<siniriGecenSayisi;i++)
  {
    TespitEdilenler temp = liste.get(i);
    cikti += "\nNesne adı = " + temp.getSinif() + "\nSkor = " +
            temp.getSkor() + "\n<---------------------------->";
  }
  return cikti;
}
}
```

Şekil 4.11. TespitEdilenler sınıfı

4.3. Bazı test sonuçları



Şekil 4.22. Test örneği-1



Şekil 4.33. Test örneği-2

BÖLÜM 5. SONUÇLAR VE ÖNERİLER

Yapay zeka ve özellikle derin öğrenmeye gelecek süreçte hayatımızda çok daha sık rastlayacağız. Ben bu proje sonucunda konvolüsyonel yapay sinir ağları ile derin öğrenme gerçekleştirilerek ilgi çekici, işe yarar, zaman ve maliyet tasarrufu sağlayacak projeler geliştirebileceğini kavradım. Burada önemli olduğunu tespit ettiğim şeyler tasarlayacağımız uygulamanın çalışacağı sistemin yeterli donanım özelliklerine sahip olması, elimizde projede çözmek istediğimiz soruna uygun bir model olması ve tabiki bunu eğitecek donanım gücü, süre ve en önemlisi verisetine sahip olmamızdır. Uygulama platformu mobil platform gibi donanım gücü daha limitli olanlara tavsiyem daha hafif modeller kullanması ve kullandığı kütüphanenin eğer varsa düşük donanımlı sistemler için tasarlanmış halini kullanmasıdır. Ayrıca yapay zeka/derin öğrenme işlemlerinin gerçekleştirildiği kısımların makineye daha yakın, hızlı dillerle yazılması büyük performans artışı sağlayacaktır. Örnek olarak bu projede yaptığım uygulamada Tensorflow kütüphanesi ile iletişime geçtiğim kısımlar C++ dili ile yazılabilirdi. Android Studio Native dil desteği sayesinde bize bu imkanı vermektedir. Fakat bunun için native kod yazmak hakkında ön bilgi gereklidir. Tabiki C++ dilini de yeterli seviyede bilmek gereklidir.

KAYNAKLAR

[1] YAPAY ZEKA UYGULAMALARI, Prof.Dr. Çetin Elmas, 3.Baskı, Sayfa 21.

[2] YAPAY ZEKA UYGULAMALARI, Prof.Dr. Çetin Elmas, 3.Baskı, Sayfa 22.

[3] http://semihsener.com/yapay-zeka-makine-ogrenimi-ve-derin-ogrenme-arasindaki-farklar/

[4] https://www.algoritmauzmani.com/nedir/bilgisayarli-gorme-computer-vision-nedir/

[5] https://medium.com/@ayyucekizrak/deri%CC%87ne-daha-deri%CC%87ne-evri%C5%9Fimli-sinir-a%C4%9Flar%C4%B1-2813a2c8b2a9

[6] https://www.udemy.com/course/bilgisayar-gorusu/

[7] http://devnot.com/2019/tensorflow-nedir-nasil-kullanilir/

[8] http://www.google.com

ÖZGEÇMİŞ

Ahmet Yasir Akbal, 30.12.1997 de İstanbul'da doğdu. İlk, orta ve lise eğitimini Kartal'da tamamladı. 2015 yılında Medine Tayfur Sökmen Lisesi'nden mezun oldu. 2016 yılında Sakarya Üniversitesi Bilgisayar Mühendisliği Bölümü'nü kazandı. Şu an Sakarya Üniversitesi'nde öğrencilik hayatına devam etmektedir.

BSM 401 BİLGİSAYAR MÜHENDİSLİĞİ TASARIMI DEĞERLENDİRME VE SÖZLÜ SINAV TUTANAĞI

KONU : Tensorflow ve OpenCV Kütüphanesi ile Android Platformunda Nesne Tespiti
ÖĞRENCİLER (Öğrenci No/AD/SOYAD): B161210074/AHMET YASİR/AKBAL

Değerlendirme Konusu	İstenenler	Not Aralığı	Not
Yazılı Çalışma			
Çalışma klavuza uygun olarak hazırlanmış mı?	x	0-5	
Teknik Yönden			
Problemin tanımı yapılmış mı?	x	0-5	
Geliştirilecek yazılımın/donanımın mimarisini içeren blok şeması (yazılımlar için veri akış şeması (dfd) da olabilir) çizilerek açıklanmış mı?			
Blok şemadaki birimler arasındaki bilgi akışına ait model/gösterim var mı?			
Yazılımın gereksinim listesi oluşturulmuş mu?			
Kullanılan/kullanılması düşünülen araçlar/teknolojiler anlatılmış mı?			
Donanımların programlanması/konfigürasyonu için yazılım gereksinimleri belirtilmiş mi?			
UML ile modelleme yapılmış mı?			
Veritabanları kullanılmış ise kavramsal model çıkarılmış mı? (Varlık ilişki modeli, noSQL kavramsal modelleri v.b.)			
Projeye yönelik iş-zaman çizelgesi çıkarılarak maliyet analizi yapılmış mı?			
Donanım bileşenlerinin maliyet analizi (prototip-adetli seri üretim vb.) çıkarılmış mı?			
Donanım için gerekli enerji analizi (minimum-uyku-aktif-maksimum) yapılmış mı?			
Grup çalışmalarında grup üyelerinin görev tanımları verilmiş mi (iş-zaman çizelgesinde belirtilebilir)?			
Sürüm denetim sistemi (Version Control System; Git, Subversion v.s.) kullanılmış mı?			
Sistemin genel testi için uygulanan metotlar ve iyileştirme süreçlerinin dökümü verilmiş mi?			
Yazılımın sızma testi yapılmış mı?			
Performans testi yapılmış mı?			
Tasarımın uygulamasında ortaya çıkan uyumsuzluklar ve aksaklıklar belirtilerek çözüm yöntemleri tartışılmış mı?			
Yapılan işlerin zorluk derecesi?	x	0-25	
Sözlü Sınav			
Yapılan sunum başarılı mı?	x	0-5	
Soruları yanıtlama yetkinliği?	x	0-20	
Devam Durumu			
Öğrenci dönem içerisindeki raporlarını düzenli olarak hazırladı mı?	x	0-5	
Diğer Maddeler			
Toplam			

DANIŞMAN : DR.ÖĞR.ÜYESİ MUHAMMED FATİH ADAK
DANIŞMAN İMZASI: