T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME İLE İŞ GÜVENLİĞİNDE KORUYUCU GÖZLÜK TESPİTİ

YÜKSEK LİSANS TEZİ

Nimetullah NECMETTİN

Enstitü Anabilim Dalı	**:**	**BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ**
Tez Danışmanı	**:**	**Dr. Öğr. Üyesi M. Fatih ADAK**

Haziran 2022

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME İLE İŞ GÜVENLİĞİNDE KORUYUCU GÖZLÜK TESPİTİ

YÜKSEK LİSANS TEZİ

Nimetullah NECMETTİN

Enstitü Anabilim Dalı : BİLGİSAYAR VE BİLİŞİM MÜHENDİSLİĞİ

Bu tez 16/06/2022 tarihinde aşağıdaki jüri tarafından oybirliği ile kabul edilmiştir.

Dr. Öğr. Üyesi	**Doç. Dr.**	**Dr. Öğr. Üyesi**
M. Fatih ADAK	**Devrim AKGÜN**	**Fahrettin HORASAN**
Jüri Başkanı	**Üye**	**Üye**

BEYAN

Tez içindeki tüm verilerin akademik kurallar çerçevesinde tarafımdan elde edildiğini, görsel ve yazılı tüm bilgi ve sonuçların akademik ve etik kurallara uygun şekilde sunulduğunu, kullanılan verilerde herhangi bir tahrifat yapılmadığını, başkalarının eserlerinden yararlanılması durumunda bilimsel normlara uygun olarak atıfta bulunulduğunu, tezde yer alan verilerin bu üniversite veya başka bir üniversitede herhangi bir tez çalışmasında kullanılmadığını beyan ederim

Nimetullah NECMETTİN
06.05.2022

TEŞEKKÜR

Türkiye'deki öğrenim süresince bana maddi ve manavı desteklerini esirgemeyen Doğu Türkistan'da Çinin zulüm kamplarına alınmış annem ve babama teşekkür borçluyum. Sizlere okuldan mezun olduğumu gösterebileceğim günlerin gelmesini diliyorum ve aynı zamanda eğitimim boyunca bana desteklerini hiç eksik etmeyen değerli eşim Sofie TURDİ'ye teşekkür ediyorum.

Yüksek lisans eğitimim boyunca değerli bilgi ve deneyimlerinden yararlandığım, her konuda bilgi ve desteğini almaktan çekinmediğim, araştırmanın planlanmasından yazılmasına kadar tüm aşamalarında yardımlarını esirgemeyen, teşvik eden, aynı titizlikte beni yönlendiren değerli danışman hocam Dr. M. Fatih ADAK'A teşekkürlerimi sunarım.

İÇİNDEKİLER

SİMGELER VE KISALTMALAR

CNN	: Evrişimsel Sinir Ağı
CSPNet	: Çapraz Aşamalı Kısmi Sinir Ağı
DenseNet	: Yoğun Evrişimsel Sinir Ağı
DL	: Derin Öğrenme
DNN	: Derin Sinir Ağı
Fast R-CNN	: Hızlı Bölge Tabanlı Evrişimsel Sinir Ağı
Faster R-CNN	: Daha-hızlı Bölge Tabanlı Evrişimsel Sinir Ağı
FPNet	: Özellik Piramit Ağı
GPU	: Grafik İşlem Ünitesi
ImageNet	: Büyük Ölçekli Görsel Tanıma Yarışması
IoU	: Intersection Over Union
mAP	: Mean Average Precision
PANet	: Yol Toplama Sinir Ağı
SPPNet	: Mekansal Piramit Havuzlama Sinir Ağı
SVM	: Destek Vektör Makinelerine
YOLO	: You Only Look Once Algoritması
YSA	: Yapay Sinir Ağı

ŞEKİLLER LİSTESİ

TABLOLAR LİSTESİ

ÖZET

Anahtar kelimeler: Göz Tespiti, Göz Koruyucu Kontrolü, Derin Öğrenme (DL), Evrişimsel Sinir Ağları (CNN), Nesne Tespiti, İş Sağlığı, YOLOv4, Koruyucu Gözlük Tespiti.

Derin öğrenme çalışmalarındaki gelişmeler ve kullanımının daha kolay olması ve bilgisayar görme teknolojilerinin gelişmesiyle görüntülerden nesne tespiti yapma, gerçek zamanlı nesne algılama sistemleri önemli bir biçimde yaygınlaşmıştır. Nesne tespiti çalışmalarından görüntülerden göz tespiti yapma işlemleri güvelik, bankacılık, adliye, tıbbi alanlar, sürücüsüz araç sistemleri, iş güvenliği ve sağlığı için çok önemli işlemler haline gelmiştir.

Bu tezde, görüntülerden gerçek zamanlı olarak göz tespiti yapan ve işletmelerde iş güvenliği ve sağlığı açsından göz koruyucu kontrolü ile koruyucu gözlük tespiti yapan bir çalışma yapılmıştır. Bu çalışmada iki farklı tespit modeli geliştirilmiş olup ilk modelde yüz uzuvlarından gözün tespiti yapılmıştır ve ikinci modelde işyerlerindeki çalışanların koruyucu gözlük kullanımını kontrol eden model geliştirilmiştir. Bu model normal gözlük ile koruyucu gözlüğü görüntülerden ayrıt edebilmektedir.

Çalışmada, internet üzerinden çalışmaya özgü elde edilmiş veri setleri ile bilgisayar üzerinde konfigürasyonu yapılmış grafik işleme ünitesi (GPU) ile eğitim işlemleri yapılarak modeller oluşturulmuştur. Modeller farklı görüntüler ile test edilerek modelin değişik ağırlıklardaki performansları karşılaştırılmıştır ve testten elde edilen sonuçların analizi yapılmıştır. Elde edilen sonuçlar CNN ağlarının hibrit kullanımı ve YOLO algoritmasının uzuv ve koruma gözlük tespitinde başarılı sonuçlar verdiği görülmüştür.

PROTECTIVE GLASSES DETECTION IN OCCUPATIONAL SAFETY WITH DEEP LEARNING

SUMMARY

Keywords: Eye Detection, Eye Protection Control, Deep Learning (DL), Convolutional Neural Networks (CNN), Object Detection, Occupational Health, YOLOv4, Safety Glasses Detection.

With developments in Deep Learning studies, more accessible use, and the development of computer vision technologies, real-time object detection systems have become widespread. Eye detection from images from object detection studies has become a significant operation for security, banking, courthouse, medical fields, driverless vehicle systems, occupational safety, and health.

In this thesis, a study was carried out that detects eyes in real-time from images and detects eye protection and protective glasses in terms of occupational safety and health in enterprises. In this study, two different detection models were developed. In the first model, the eye was detected from the facial parts. In the second model, a model was developed to control the use of protective glasses by the employees in the workplace. This model can distinguish regular glasses and protective glasses from images.

In the study, models were created by training with the graphics processing unit (GPU) configured on the computer with the datasets obtained specifically for working over the internet. The models were tested with different images, the performances of the model at different weights were compared, and the results obtained with the test were analyzed. The results obtained showed that the hybrid use of CNN networks and the YOLO algorithm gave successful results in the detection of limbs and goggles.

BÖLÜM 1. GİRİŞ

Dünya genelinde verilere erişimin kolaylığı ve bu verileri işlemek için gerekli yazılımların gelişmesi, daha yaygın kullanılması ve donanımlardaki hızlı gelişmelerden dolayı Derin Öğrenme (DL) daha yaygın kullanılmaya başladı. Son yıllarda en çok duyduğumuz, günlük hayatımızdan tutun sanayilere ve hatta uzay teknolojilerine kadar adından bahsettiren bir teknoloji olmuştur. Derin öğrenme ile günlük hayatımızdaki en ufak problemlerden tutun uzay bilimlerine kadar karşılaştığımız problemleri hızlı ve yüksek başarım oranları ile çözümüne kavuşturmaktadır. Bugünlerde görüntü sınıflandırma, video analizi, doğal dil işleme, ses tanıma, borsa fiyat tahminleri, medikal tedavi sanal gerçeklik (VR), arttırılmış gerçeklik ve daha çok alanlardaki problemlere üniversiteden özel firmalara kadar derin öğrenme yöntemleri ile çözüm üretmektedir [1], [3].

Göz, insan yüzündeki burun veya ağız gibi diğer özelliklere karşılaştırıldığında daha belirgin bir özelliktir. Bu nedenle, göz tespiti multimedya cihazları için insan-makine ara yüzü oluşturma, insan-makine iş birliği, insan-bilgisayar etkileşimi, bakış takibi, psikolojik analiz, akıllı araç sistemleri için sürücü yorgunluk analizi, sürücü davranışlar analizi, adli bilişim, otomattık yüz algılama ve tanıma gibi çeşitli sistemler için geniş çapta çalışılmıştır [1], [2], [3].

Yüz görüntülerinde göz algılama, yüz tanıma, bakış takibi, bilgi güvenliği, bankacılık işlemleri, adliye, sağlık ve dijital oyun gibi günlük hayatımızla ilgili her alanlarda her zaman önemli bir yere sahiptir. Görüntü işleme metotlarının gelişmesiyle birlikte, çoğu tanımlama sistemi artık biyometrik özelliklere dayanmaktadır [2], [3]. Yüz tanıma, son birkaç yılda çok ilgi gördü ve görüntü analizi alanında en umut verici uygulamalardan biri olarak kabul ediliyor. Yüz algılama, yüz tanıma işlemlerinin büyük bir bölümünü kapsayabilir. Resimlerdeki yüz algılama yöntemleri, poz, yüz

ifadesi, konum ve yön, ten rengi, gözlük veya sakal, aydınlatma koşulları ve görüntü çözünürlüğü gibi değişkenlik nedeniyle karmaşıktır. Ancak pandemi nedeniyle insanlar sürekli maske veya koruyucu giysiler takması yüz tanımlama işlemlerini zorlaştırıyor. Bu nedenle, tanımlama görevlerinde mevcut biyometrik özelliklerin yerine insan kulağı, gözü ve burnu gibi uzuvların kullanılması da oldukça önemlidir.

İşletmelerde koruyu gözlük kullanımının denetlenmesinde göz tespitinin yapılması önemli bir işlemdir. Göz tespit yöntemi ile iş sağlığı açısından çalışanların koruyucu gözlük kullanılması zorunlu olan iş yerlerinde çalışanların koruyu gözlük kullanımını denetlenmesinin yapılması denetleyici personellere ihtiyaç olmadan göz ya da koruyucu gözlük kullanımın kontrol eden modeller ile yapılabilmesi mümkündür. Bu işletmeler açısından denetlemek için personele olan ihtiyacı ortadan kaldırır, işletmelerin iş yükünü azaltır ve gerçek zamanlı olarak çalışanları sağlıklı bir ortamda çalışmasını denetler.

1.1. Literatür Özeti

Yüz ve yüz uzuv tespiti, nesne tanıma alanındaki zorlu problemlerden biridir. Değişik derin öğrenme ve gerçek zamanlı nesne algılama yöntemleri ile yapılan yüz ve yüzdeki diğer uzuvların tespiti hakkındaki çalışmalar literatürde mevcuttur. Yapay zekâ kullanılarak nesne tespiti yapan iki basamaklı (Two Stages) Fast_CNN, Faster_CNN gibi nesne algılama algoritmaları ve tek basamaklı YOLO gibi gerçek zamanlı nesne algılama algoritması ile daha hızlı ve çok yüksek performanslı sergileyen çalışmalar ve yöntemler önerilmektedir.

Yu M. vd. Destek Vektör Makinelerine (SVM) ve gri yoğunluk Varyans Filtresine (VF) dayalı hibrit göz algılama yöntemi önermişlerdir. Göz bölgelerindeki gri yoğunluk değişimi diğer bölgelerden daha belirgin olduğundan, ilk olarak, VF ile göz olmayan bölgeler tespit edip ortadan kaldırmıştır. Daha sonra kesin göz konumlarını belirlemek için eğitilmiş SVM sınıflandırıcı ile iki göz bölgesi kolayca belirlenmiştir. SVM'nin çalışma süresini azaltmak ve sınıflandırma doğruluğunu artırmak için, veri boyutunu azaltma ve öznitelik çıkartmada PCA ve SVM parametrelerini optimize

etmek için GA kullanılmıştır. Sonuçlar, önerilen yöntemin yeterince sağlam olduğunu göstermektedir [3].

Karahan S. vd. tarafından 2016 yılında yapılan çalışmada, derin öğrenme ile göz tespiti yapılmıştır. Bu çalışmada, derin öğrenmede sıklıkla kullanılan Caffe [4] kütüphanesi DIGITS [5] web tabanlı programı üzerinden modelin eğitimi sağlanmıştır. Modelde birkaç katmandan oluşan evrişimsel yapay sınır ağları kullanarak model oluşturulmuştur. Bu model ile Haar algoritması FDDB [6] ve CACD [7] veri kümelerinden tespit edilen yüz imgeleri veri seti ile test edilerek kesinlik ve hassasıyet değerleri hesaplanmıştır. Geliştirilen modelin çağırışım değerleri her iki veri kümesinde iyi sonuçlar verirken kesinlik değeri de Haar algoritmasına göre FDDB veri kümesinde daha düşük performans göstermiş ve tespiti zor veri kümelerinde geliştirilen modelin Haar'a göre daha iyi olduğu sonuçlardan gözlemlenmiştir [1].

Donuk K. vd. Evrişimsel Sınır Ağları (CNN) modeli kullanarak göz tespitinde yüksek başarım oranı elde etmiştir. Çalışmada "Large-scale CelebFaces Attributes (CelebA)" ve "Closed Eyes In The Wild (CEW)" veri setlerindeki değişik poz, farklı aydınlatma koşulları, gözlüklü ve gözlüksüz imgelerden oluşan 260 bine yakın imgeler model eğitiminde kullanılmış. Verilerin %20'lık bölümü test aşamasında geri kalan %80'lık bölümü ise eğitim aşamasında kullanılmış. Önerilen yöntemin eğitim aşaması %99 ve test aşaması %97,6'lık başarım oranı ile sonuçlanmış olup Viola-Jones algoritması göre daha iyi sonuçlar verdiği görülmüştür [2].

Civik E. vd., Sürücünin yorgunluk durumunu tespit etmek için gerçek zamanlı gömülü sistemli derin öğrenme bazlı bir sistem geliştirmiş olup, sistem sürücünün göz ve ağızlarını tespit ederek sürücünün yorğunluk durumuna göre ikaz edebilmektedir. Bu çalışmada, VGG-16 ağı ile eğitilmiş 2 model kullanılmış ve veri seti olarak YawDD kullanılmıştır. Bu veriler modele uygulanmadan önce OpenCV ile ön işlemden geçirilerek yeniden boyutlandırılmıştır. Test veri setinden göz tespit modeli için 674, ağız tespit modeli için ise 623 video görüntüsü kullanılmıştır. Sistemin doğruluğu %94,05 olarak hesaplanmış olup göz modelinin doğruluğu %93,6 iken, ağız modelinin doğruluğu %94,5 olarak bulunmuştur. Dünya ve ülkemiz genelinde bakıldığında,

trafik kazalarının yarattığı maddi ve manevi zararlar, bunların toplumsal bir problem olduğunu göstermekledir [9]. TÜİK'in [10] 2021 yıllık raporuna göre, 2020 yılında, Türkiye'de meydana gelen trafik kazalarında hayatını kaybedenlerin sayısı 4866 ve kazalara neden olan kusurlardan sürücü kusurları %88,3'ü oluşturarak ilk sıralarda yer almaktadır. Yapılan analizler ve çalışmalarda gerçek zamanlı bir sürücü göz durum tespiti sistemi ile bu kazaların önlenebileceği ortaya çıkarmaktadır.

Nguyen Quoc. vd. yeni bir kulak algılama sistemi önermiştir. Sistem YOLOV3 ve RetinaFace tabanlı olup, sistemin gerçek zamanlı uygulamalara uyum sağlayabilmesi için çok sayıda değişiklik eklenmiştir. Deney sonuçları hem çıkarım hızı hem de doğruluk açısından önceki kulak tespit sistemleri göre daha iyi performans göstermiştir [11].

Aung. vd. çalışmasında, yüz algılama sistemleri için bir iyileştirme önermek için YOLO (Yalnızca Bir Kez Bakarsınız) algoritması ve VGG16 modeli ile bir hybrid model oluşturmuştur. Bu modelde, önceden eğitilmiş VGG16 modelinin son 9 katmanı kaldırılmış ve modelin çıktısı 9 katmandan oluşan YOLO nesne algılama ağına eklenerek oluşturulmuştur. Sonuçlardan, önerilen yöntemin test görüntüsünü %95'in üzerinde ortalama hassasiyetle tespit ettiği, ayrıca, gerçek zamanlı canlı videoda yüz algılama hızını önemli ölçüde artırdığı gözlemlenmiştir [12].

Garg. vd. [13] derin öğrenme yöntemiyle yüz tespiti çalışması yapmıştır. Önerilen modelin mimari konvolüsyon katmanı, maksimum havuzlama katmanı izler, tam bağlı katman ve YOLO katmanından oluşturulmuş. Konvolüsyon katmanları yüz resimlerinden özellik çıkartmada ve tam bağlantılı katman, koordinatları ve olasılıkları tahmininde kullanılmış. Son olarak, çıktı katmanı, NMS (Maksimum Olmayan Bastırma) tekniğini kullanarak hem sınıf olasılıklarını hem de sınırlayıcı kutunun koordinatlarını tahmin etmiştir.

Göz hastalıklarının erken tespitinde [14], göz bölgesinin yerinin belirlenmesi tanı için çok önemlidir. Bu araştırmada, Kızılötesi termal görüntülerinden gözün yerini tespitinde ResNet50 gözün özelliklerini ve olasılık oranını bulmak için kullanılmış ve

YOLOV2'da göz bölgesinin otomatik olarak tespit edilmesini yüksek doğruluk ve yüksek hız ile sağlamaktadır. Sonuçlardan önerilen modelin göz bölgesinin lokalizasyonu için diğer uzman destekli modellere göre yüksek bir performans sergilediği gözlemlenmektedir.

Literatür taramasında yapılmış çalışmaların kullandığı teknikler, veri seti ve doğruluk oranı gibi bilgiler Tablo 1.1.'de gösterilmiştir.

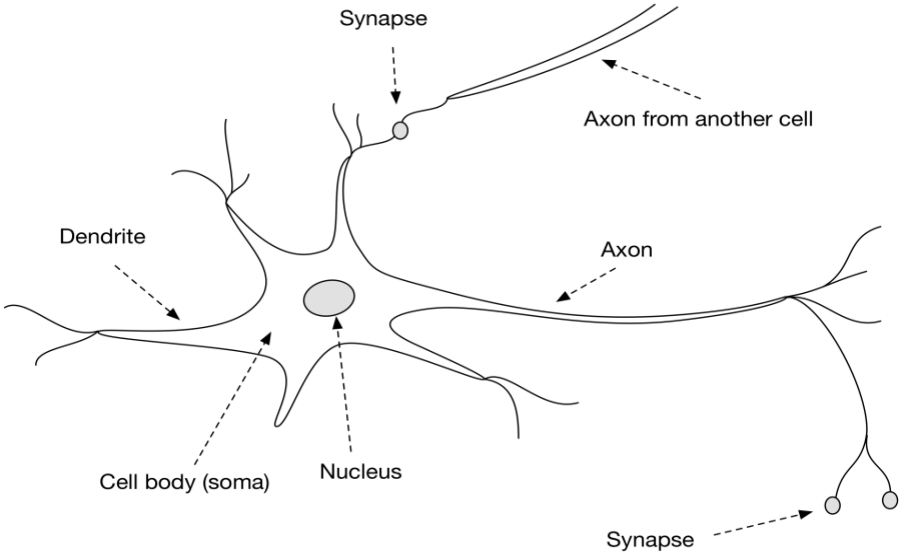
Tablo 1.1. Literatür taramasında yapılmiş çalışmaların listesi

No	Yapılan Uygulama	Kullanılan Teknikler	Kullanılan Veri seti	Yazarlar	Doğruluk
[1]	Göz tespiti	Evrişimsel yapay sınır ağı, Haar algoritması	FDDB [6] , CACD [7]	Karahan vd. 2016	%94
[2]	Yüz İmgelerinden Göz Bölgelerinin Tespiti	Evrişimsel Sınır Ağları (CNN)	Large-scale CelebFaces Attributes (CelebA), Closed Eyes In The Wild (CEW)	Donuk vd. 2021	%97,6
[3]	Göz tespiti	Destek Vektör Makinelerine (SVM) ve gri yoğunluk Varyans Filtresi (VF), PCA ve GA algoritması	Biometric Identity (BioID)	Yu vd. 2016	%95,6
[9]	Sürücülerin yorgunluk durumunu tespit, Göz tespiti	VGG16	YawDD	Civik vd. 2020	%94,05
[11]	Kulak tespiti	YOLOV3 ve RetinaFace tabanlı	Asyalı ünlülerin resimlerinden rastgele oluşturulan veri seti	Nguyen Quoc vd. 2021	%71,2
[12]	Yüz algılama	YOLO, VGG16	FDDB [6]	Aung vd. 2021	%95
[13]	Yüz algılama	Konvolüsyon sınır ağı, YOLO	FDDB [6]	Garg vd. 2018	%92,2
[14]	Göz tespiti	ResNet50, YOLOv2	Department of Ophthalmology, Tagore Medical College Hospital'da kamera ile alınmış göz görüntüleri	Madura Meenakshi vd. 2021	%97

BÖLÜM 2. GÖZ TESPİTİNDE KULLANILAN TEKNOLOJİLER

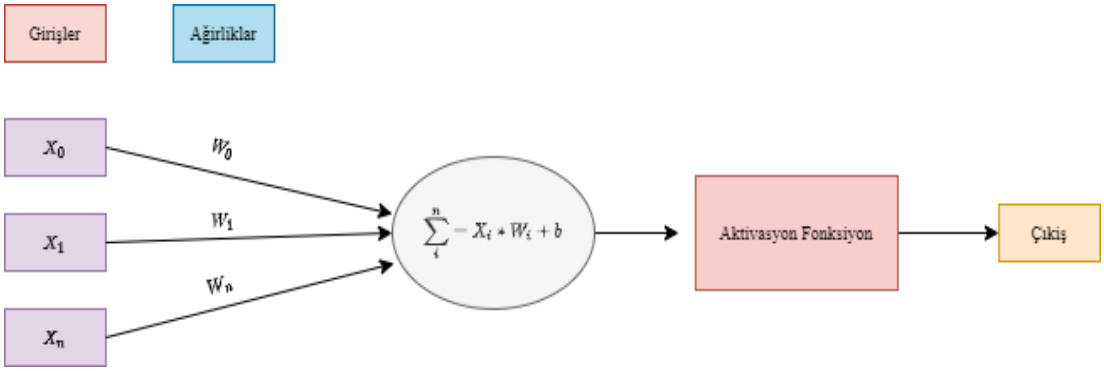
2.1. Yapay Sınır Ağları (YSA)

Yapay sinir ağı (YSA), makine öğrenmesi ve bilişsel bilim alanında sinir ağı olarak anılan, bilgi işleme için biyolojik sinir ağlarını taklit eden matematiksel bir modeldir. Beynin fizyolojik araştırma sonuçlarına dayanmaktadır ve amacı, bazı belirli işlevleri gerçekleştirmek için beynin bazı mekanizmalarını taklit etmektir. Şekil 2.1.'de nöronu oluşturan hücreleri göstermektedir. Sinir ağları, çok sayıda yapay nöronun bağlanmasıyla hesaplanır. Çoğu durumda yapay sinir ağları, dış bilgi temelinde iç yapıyı değiştirebilir. Modern sinir ağı bir tür doğrusal olmayan istatistiksel veri modellemedir. Sinir ağı genellikle matematiksel istatistik türüne dayalı bir öğrenme yöntemi ile optimize edilir, bu nedenle aynı zamanda matematiksel istatistik yönteminin pratik bir uygulamasıdır. Öte yandan, yapay zekadaki yapay algı alanında, yapay algı hakkında kararlar vermek için matematiksel istatistikleri kullanabiliriz [16].



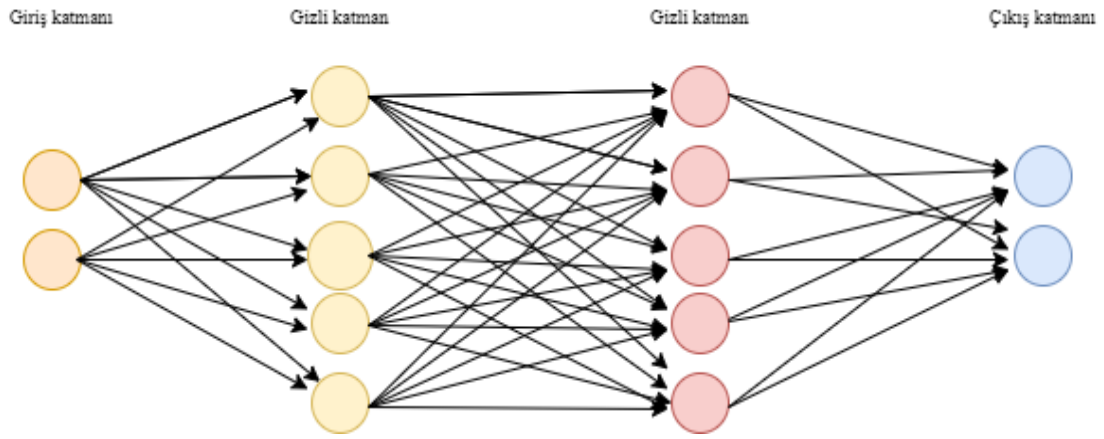
Şekil 2.1. Biyolojik nöron [16]

Şekil 2.2.'de YSA'daki bir nöronun matematiksel modelini göstermektedir. Yapay nöron aldığı tüm giriş değerini nöron hücrelerindeki ağırlıkla çarpıldıktan sonra bir bias ile toplama işlemi yapar ve bunu bir aktivasyon fonksiyonundan geçirdikten sonra çıkış katmanına aktarır.



Şekil 2.2. Yapay nöron

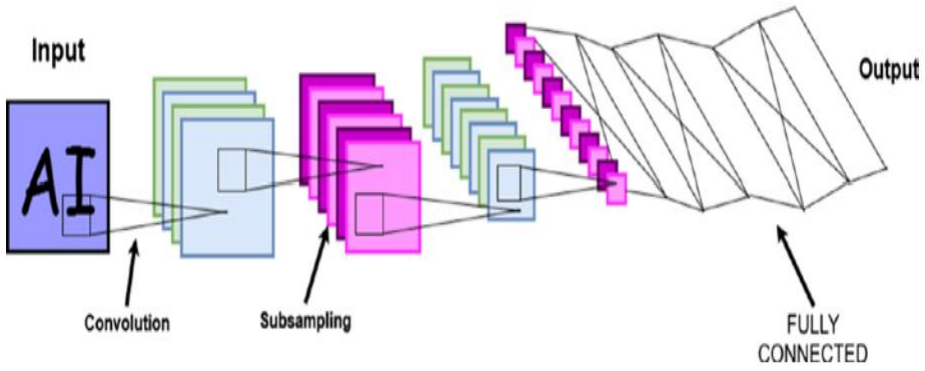
YSA'da tıpkı insan sinir sistemi gibi birden fazla nöronların bir araya gelmesi ile oluşmaktadır. YSA girdi katmanı, kullanım ihtiyaçlarına göre tek katman ya da birden çok katmandan oluşan gizli katmanı ve son olarak da bir çıktı katmanından oluşmaktadır. Şekil 2.3.'de birçok katmanlı YSA gösterilmektedir. Yapay nöronun çalışma modelinden de görüldüğü gibi girdiler ağın ağırlığı ile çarpılarak bir sonraki katmana iletilir. ReLU, Sızıntı ReLU, Sigmoid, Tanh, Swish, Softmax gibi aktivasyon fonksiyonundan geçen bu değerler çıktı katmanına ve bir sonraki katman için girdi değerleri olarak iletilir [17].



Şekil 2.3. Çok katmanlı yapay sınır ağı

2.2. Evrişimsel Sinir Ağları (CNN) ve Mimarisi

Evrişimsel sinir ağları, özellikle umut verici bir derin öğrenme biçimi olarak son birkaç yılda özel bir önem kazanmıştır. Görüntü işlemeye dayanan evrişimsel katmanlar, derin öğrenmenin neredeyse tüm alt alanlarına girmenin yolunu bulmuş ve çoğunlukla çok başarılıdır. CNN'ler özünde, evrişim işlemini katmanlarından biri olarak kullanan sinir ağlarıdır. CNN farklı filtreleme işlemleri (Şekil 2.4.) sayesinde görüntülerden tahmin için gerekli özellikleri elde ediyor ve birbirinden farklı fonksiyonlara sahip evrişim, ortaklama ve tam bağlı katmanlardan oluşmaktadır. CNN'ler, tahminlerin yapılacağı girdi verilerinin bir görüntü gibi bilinen bir ızgara benzeri topolojiye sahip olduğu problemlere uygulanan inanılmaz derecede başarılı bir teknolojidir [18].

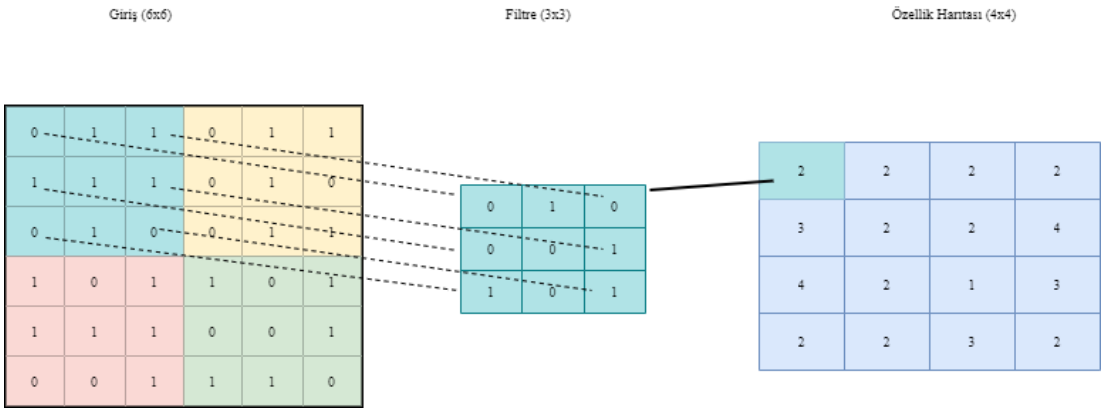


Şekil 2.4. CNN'deki katmanlar [19]

2.2.1. Evrişim katmanı (Convolutional Layer)

Evrişim katmanı, filtre olarak adlandırılan matrisi giriş görüntüsü üzerinde satır satır kaydırır ve bu işlemin sayesinde görüntüdeki özellikleri çıkarır. Evrişim işleminin işlevi, geleneksel görüntü işlemedeki filtrelemeye benzerdir. Genel olarak, bir evrişim temel katmanı esasen bir matristir, genellikle 3x3, 5x5 veya 7x7 gibi farklı boyutlardan oluşmaktadır. Evrişim işleminin adım aralığı (Stride) kayan pencerenin görüntü üzerinde kaç adımla kaydırma yapılacağını belirlemektedir. Bu filtredeki ağırlık matrisi her seferinde giriş görüntüsündeki pikseller üzerinde kaydırılır ve bu filtreleme işlemi sırasında filtrene matrisi ile görüntü üzerindeki filtre matrisi boyutundaki

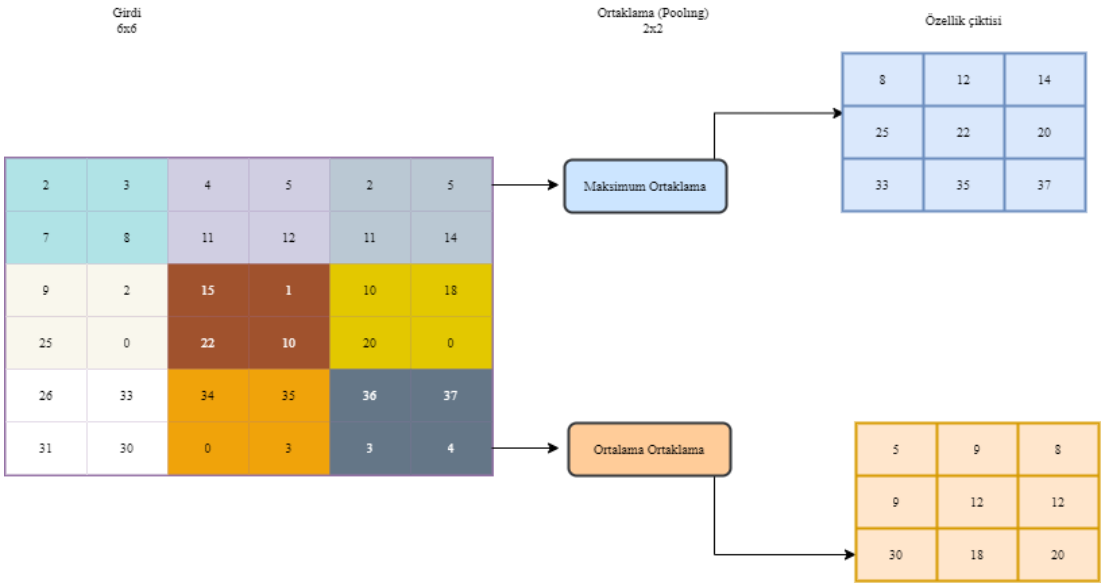
değerle çarptırılarak görüntünün özellik haritası çıkarılır (Şekil 2.5.). Ağdaki filtrelerin sayısı ne kadar çok olursa o kadar çok özellik haritası çıkarılabilir ve görüntüdeki kalıpların öğrenilmesi kolaylaşır [20].



Şekil 2.5. Evrişim işlemi

2.2.2. Havuzlama katmanı (Pooling Layer)

Havuzlama, genelde evrişim katmanından sonra uygulanan ve giriş boyutlarında küçültme işlemi gören bir işleve sahip katmandır. Bu katman maksimum havuzlama ve ortalama havuzlama gibi iki türleri vardır. Evrişimsel katman gibi maksimum havuzlama katmanında da filtreler mevcuttur ve bu filtreler görüntü üzerinde kaydırılırken o penceredeki değerlerden en büyük değeri yanı maksimum değeri seçer. Ortalama havuzlama ise penceredeki değerlerin ortalama değerini çıkararak bir sonraki katmana iletir (Şekil 2.6.). Böylece görüntünün yükseklik ve genişliğinde küçülmeye gider. Bu işlem ağın hesaplama yükünü hafifletir ve ağın daha hızlı çalışmasını sağlar. Aynı zamanda ağdaki aşırı öğrenme problemini de önler ama boyutlarda küçülme olduğu için bilgi kaybına da yol açabilir [20].



Şekil 2.6. Ortaklama katmanındaki işlemler

2.2.3. Tam bağlantılı katman (Fully Connected Layer)

Temel CNN ağında, tam bağlantılı katmanın işlevi, görüntü özelliklerinin üst düzey anlamını elde etmek için çoklu evrişim katmanlarından ve havuzlama katmanlarından sonra görüntü özellik haritasındaki özellikleri entegre etmek ve ardından bunları görüntü sınıflandırması için kullanmaktır. Bir CNN ağında, tam bağlantılı katman, evrişim katmanı tarafından oluşturulan özellik haritasını sabit uzunlukta bir özellik vektörüne eşler. Bu özellik vektörü, giriş görüntüsünün tüm özelliklerinin birleşik bilgisini içerir. Görüntünün konum bilgisi kaybolmasına rağmen, vektör görüntü sınıflandırma görevini tamamlamak için görüntüdeki en karakteristik görüntü özelliklerini korur. Görüntü sınıflandırma görevi açısından, bilgisayarın yalnızca görüntü içeriğini belirlemesi, girdi görüntüsünün belirli kategori değerini hesaplaması ve sınıflandırma görevini tamamlamak için en olası kategoriyi çıkarması gerekir [21].

Tam Bağlantılı Katman genellikle tüm evrişimli sinir ağının sonunda bulunur ve evrişim tarafından iki boyutlu özellik haritası çıktısını tek boyutlu bir vektöre dönüştürmekten, böylece uçtan uca öğrenme sürecini gerçekleştirmekten sorumludur. Tam bağlı katmanın her düğümü, önceki katmanın tüm düğümlerine bağlıdır, bu nedenle buna tam bağlı katman denir. Tam bağlantılı katmanın ana işlevi, önceki katman tarafından hesaplanan özellik uzayını örnek etiket alanına eşlemektir. Basitçe

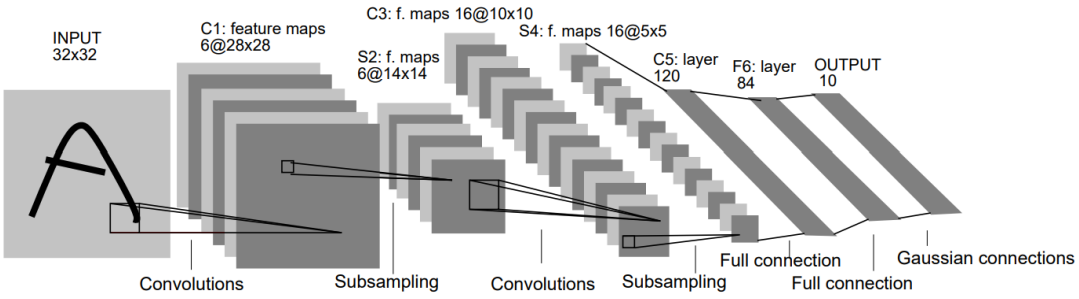
söylemek gerekirse, özellik konumunun sınıflandırma sonuçları üzerindeki etkisini azaltma ve tüm ağın sağlamlığını artırma avantajına sahip olan özellik temsilini tek bir değerde entegre etmektir [22].

2.3. Evrişimsel Sinir Ağlarındaki Farklı Mimariler

Derin öğrenme, makine öğrenmesi ve yapay zekâ araştırmalarındaki popüler konu başlıklarından biridir. Son günlerde hızla gelişen yeni bir alan olarak, araştırmacıların giderek daha fazla ilgisini çekmektedir. Evrişimsel Sinir Ağı (CNN) modeli, derin öğrenme modellerinde en önemli klasik yapılardan biridir ve performansı son yıllarda derin öğrenme görevlerinde giderek artmıştır. Evrişimli sinir ağları, örnek verilerin özellik gösterimini otomatik olarak öğrenebildikleri için görüntü sınıflandırma, nesne algılama, anlamsal bölümleme ve doğal dil işlemede yaygın olarak kullanılmaktadır. Evrişimli sinir ağının gücü, çok katmanlı ağ yapısının giriş verilerinin derin özelliklerini otomatik olarak öğrenebilmesi ve farklı seviyelerdeki ağların farklı özellik seviyelerini öğrenebilmesidir. CNN'leri temel alan farklı katman, parametreler ve işlevlere sahip gelişmiş modeller gün geçtikçe daha iyi performans ve doğruluk oranları göstermektedir.

2.3.1. LeNet

1998 yılında, LeCun ve diğerleri, derin öğrenmenin perdesini açan LeNet ağını piyasaya sürdü ve ardından gelen derin sinir ağları bu temelde geliştirildi ve yapısı Şekil 2.7.'de gösterilmektedir. Şekilde gösterildiği gibi, LeNet-5 evrişimli katmanların, havuzlama katmanlarının ve tam bağlı katmanların sıralı bir bağlantısından oluşur. Ağdaki her katman, verileri etkinleştirmek için türevlenebilir bir işlev kullanır. Bir katmandan diğerine geçer. LeNet-5, hesaplama performansı sınırlı olduğunda çok fazla hesaplama tasarrufu sağlayabilen doğrudan görüntülerden özellikleri öğrenmek için evrişim kullanımına öncülük etti ve ayrıca evrişimli katmanların kullanımının görüntülerin uzamsal korelasyonunu sağlayabileceğine işaret ediyor [23].



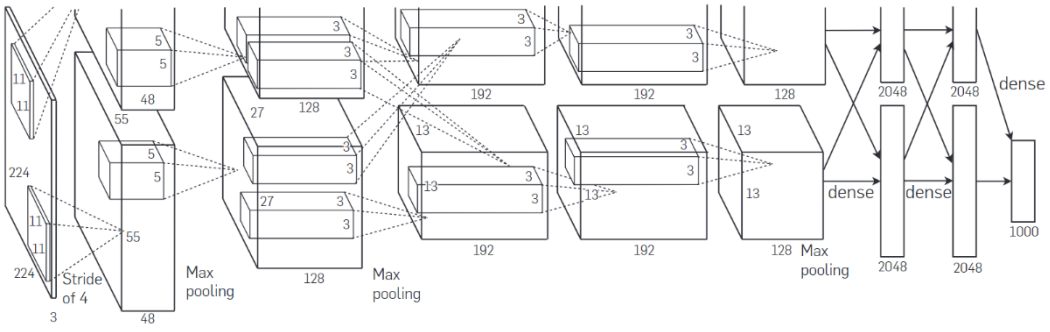
Şekil 2.7. LeNet-5 Mimarisi [23]

LeNet-5, 7 katmanlı bir CNN'den oluşmaktadır. Her katmanın anlamı aşağıda açıklanmıştır;

- Girdi: Sinir ağının girişi 32x32 gri tonlamalı bir görüntüdür.
- C1: İlk katman bir evrişim katmanıdır, evrişim çekirdeği (çekirdek boyutu) 5x5'tir, adım aralığı 1'dir ve dolgu (Padding) işlemi yapılmamaktadır, bu nedenle giriş görüntüsü bu katmandan hemen sonra 28 x28 boyutlu 6 özellik haritası çıkacaktır.
- S2: Maksimum ortaklama bir alt örnekleme katmanıdır. Adım aralığı 2'dir ve filtre boyutu 2x2'dir.
- C3: Ağdaki üçüncü evrişim katmanıdır. Çekirdek boyutu ve adım aralığı ilk katmanla aynıdır, ancak 16 özellik haritası çıkacaktır.
- S4: Dördüncü katmandır ve ağdaki ikinci ortaklama katmanıdır.
- FC5: Bir tam bağlantılı katmandır. Dördüncü katmanın özellik haritasını düzleştirir.
- FC6: Bu katman 84 birim içerir ve tamamen FC5 katmanına bağlıdır.
- Çıktı: Çıkış katmanı, sınıf başına bir birim olan Öklid Radyal Temel Fonksiyon birimlerinden oluşur ve her bir birimin 84 girişi vardır. Çıkış uzunluğu 10 olan bir tensör elde etmek için bir Softmax çoklu sınıflandırma fonksiyonu uygulanır [23].

2.3.2. AlexNet

2012'de Alex Krizhevsky ve diğerleri tarafından önerilen AlexNet, ImageNet yarışmasını büyük bir farkla kazandı. Bu başarı, endüstrinin sinir ağlarına olan ilgisini büyük ölçüde teşvik etti, görüntü problemlerini çözmek için derin sinir ağlarını kullanmanın bir yolunu açtı ve ardından bu alanda giderek daha mükemmel sonuçlar ortaya çıktı. AlexNet 5 evrişim katmanı, 2 tam bağlı gizli katman ve bir tam bağlı çıktı katmanı olmak üzere 8 katmandan oluşmaktadır. Ağın tasarımı LeNet'e çok benzemektedir ancak LeNet'e kıyasla çok daha derindir ve ağdakı hesaplama karmaşıklığını azaltmak için aktivasyon fonksiyonu olarak Tanh veya Sigmoid yerine ReLU'yu tercih ediyor. AlexNet'in ilk katmanı 11x11 boyutlu filtreye sahip evreşim katmanıdır, burada böyle büyük boyuttaki filtrenin kullanılmasını nedeni ağa giren görüntülerin boyutları MNIST verilerinin boyutlarından daha büyüktür ve böylece görüntüler üzerinde daha belirgin özellik haritası oluşturuluyor. Sonradaki evrişim katmanlarındaki filtre boyutları 5x5 ve 3x3 olarak adım adım küçülüyor. Ağdaki evrişim katmanların arasında 3x3 boyutlu adım aralığı 2 olan maksimum ortaklama katmanları da vardır. Ağın yapısı 2 adet GPU üzerinde çalışmak için uyarlanmış olup, Şekil 2.8.'de gösterilen üst katmanlar bir GPU'da ve alttaki katmanlar ise diğer GPU'da çalışmaktadır [24], [25].



Şekil 2.8. AlexNet'in katman mimarisi [24]

AlexNet, LeNet ile karşılaştırıldığında, daha derin ve geniş bir ağ yapısına sahiptir ve modelin eğitim sürecini iyileştirmek için aşağıdaki üç yöntemi kullanır:

- Veri büyütme (Data Augmentation): Derin öğrenmede yaygın olarak kullanılan veri işleme yöntemidir. Eğitime, çeviri, ölçekleme, kırpma, döndürme, çevirme veya parlaklığı artırma veya azaltma gibi bazı değişiklikleri rastgele ekleyerek, orijinaline benzeyen ancak aynı olmayan bir dizi görüntü ile eğitim veri setini genişletmek için örnekler oluşturulur.
- Dropout: Aşırı öğrenme (Overfitting) problemini önlemek için dropout yöntem yöntemi kullanılarak ağdaki nöronları azaltıyor.
- ReLU aktivasyon fonkısyonu kullanılarak Gradient kaybolma olgusunu ve hesaplama yükünü azaltıyor [24].

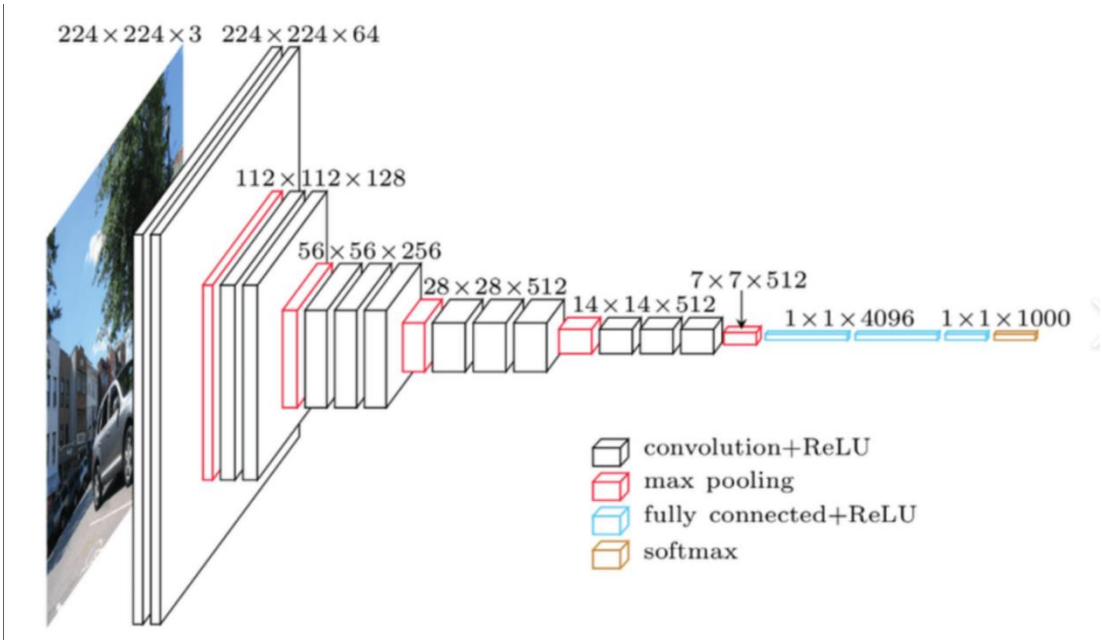
AlexNet, LeNet ile karşılaştırıldığında, daha derin ve geniş bir ağ yapısına sahiptir ve modelin eğitim sürecini iyileştirmek için aşağıdaki üç yöntemi kullanır:

- Veri büyütme (Data Augmentation): Derin öğrenmede yaygın olarak kullanılan veri işleme yöntemidir. Eğitime, çeviri, ölçekleme, kırpma, döndürme, çevirme veya parlaklığı artırma veya azaltma gibi bazı değişiklikleri rastgele ekleyerek, orijinaline benzeyen ancak aynı olmayan bir dizi görüntü ile eğitim veri setini genişletmek için örnekler oluşturulur.
- Dropout: Aşırı öğrenme (Overfitting) problemini önlemek için dropout yöntem yöntemi kullanılarak ağdaki nöronları azaltıyor.
- ReLU aktivasyon fonkısyonu kullanılarak Gradient kaybolma olgusunu ve hesaplama yükünü azaltıyor [24].

2.3.3. VGGNet

AlexNet'in ortaya çıkmasıyla beraber, derin evrişimsel ağların iyi sonuçlar alabıleceğını kanıtlamasının ardından 2014'te Simonyan ve arkadaşı VGGNet olarak adlandırılan modeli tanıttı. Bu model AlexNet ve LeNet gibi aynı evrişimsel yapılara sahıptı. Ancak evrişimsel katmandakı filtre sayısı AlexNet'e gibi farklı boyutlarda değil de sabıt 3x3 boyutlu filtrelerden oluşmaktadır. Modelin katmanları 11 ile 19 arasında değişmektedir ve esas olarak VGG-16 ve VGG-19 olarak iki farklı versiyonu çok kullanılmaktadır. Şekil 2.9.'da görüldüğü gibi evrişimsel katmanlarda ReLU

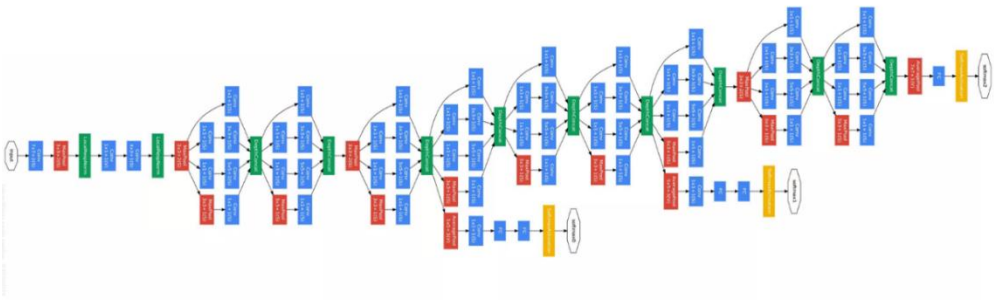
aktivasyon fonksiyonu olarak kullanılmaktadır ve çıkış katmanında Softmax aktivasyonu kullanarak çoklu görüntü sınıflandırması yapılmaktadır [26].



Şekil 2.9. VGGNet-19 Mimarisi [27]

2.3.4. GoogLeNet

2014 yılında ImageNet yarışmasının sınıflandırma görevinde GoogLeNet birinciliği kazandı. GoogLeNet, Google ekibi tarafından geliştirilen derin bir ağ yapısıdır. VGGNet modeliyle karşılaştırıldığında, GoogLeNet modelinin ağ derinliği 22 katmana ulaştı ve modelin genel performansını daha da iyileştirmek için Inception birimi ağ yapısına dahil edildi. Derinlik 22 katmana ulaşsa da boyut AlexNet ve VGGNet'den çok daha küçüktür. GoogLeNet mimarisi, AlexNet mimarisinden 12 kat daha az parametreye sahipken, doğruluğu önemli ölçüde artırılmıştır. Nesne algılamadaki en büyük kazançlar, yalnızca derin ağların veya daha büyük modellerin kullanılmasından değil, R-CNN algoritması gibi klasik bilgisayar vizyonunun sinerjisinden gelir [28].

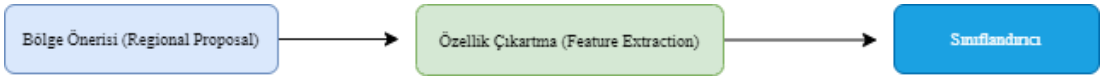


Şekil 2.10. GoogLeNet katman yapısı [28]

2.4. Nesne Algılama Modelleri

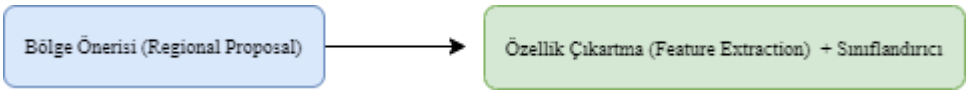
Uzun yıllar boyunca, elle çıkarılan yerel olarak değişmez özelliklerden ve ayırt edici sınıflandırıcılardan oluşan çok aşamalı modeller, 2012 yılında derin evrişimli sinir ağları (DCNN'ler) öne çıkana kadar nesne algılama dahil olmak üzere çeşitli bilgisayar görüsü alanlarına egemen oldu. Girshick ve diğerleri, DCNN'ye dayalı bir bölgesel CNN (RCNN) önerdiler ve nesne algılama alanına başarılı bir şekilde derin sinir ağını uyguladılar. O zamandan beri, derin öğrenmeye dayalı model yöntemi, insanlarla eşleşen hedef tanıma yeteneğini elde etmeye çalışan hedef tespitinin ana araştırma yönü haline geldi. Bugün, Faster RCNN ve YOLO tarafından temsil edilen mükemmel hedef tespit modelleri, yavaş yavaş eksiksiz bir görüntü tespit sistemine dönüşmüştür.

Nesne algılamada, geleneksel nesne algılama ve Bölge Önerisine (Regional Proposal) dayalı derin öğrenme tespit algoritmaları mevcuttur. Geleneksel nesne algılamada (Şekil 2.11.) kayan pencere tüm görüntüde gezinerek farklı ölçekler ve boyuttaki oranları tespit eder. Daha sonra SIFT, HOG gibi yaygın olarak kullanılan öznitelik çıkarma yöntemlerini ile hedefin morfolojik, aydınlatma değişiklikleri ve arka planların çeşitliliği kullanılarak özellik çıkarımı yapılır ve son olarak SVM, AdaBoost gibi sınıflandırıcılar ile nesne sınıflandırması yapar [29].



Şekil 2.11. Geleneksel nesne tespit adımları

Bölge Önerisine (Regional Proposal) dayalı derin öğrenme tespit algoritmalarında, bölge önerisi, görüntüdeki hedefin nerede görünebileceğini önceden bulmaktır. Bölge önerisi, görüntüdeki doku, kenar, renk ve diğer bilgileri kullandığından, daha az pencere seçildiğinde yüksek bir geri çağırma oranı sağlayabilir. Aday bölgeler elde edildikten sonra, özellik çıkarma ile sınıflandırma işlemlerinin birleşmesi olan CNN sınıflandırma modeli kullanılarak sınıflandırma işlemleri yapılıyor.



Şekil 2.12. Bölgesel öneriye dayalı derin öğrenme tabanlı nesne tespitinin adımları

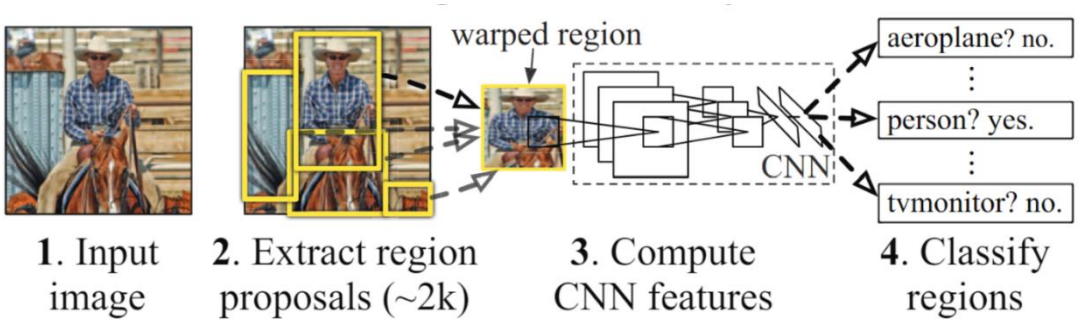
Derin öğrenmeye dayalı nesne algılama modellerinde özellikle, RCNN, SPPNET ve Faster RCNN tarafından temsil edilen iki aşamalı algılama modelleri ile YOLO ve SSD tarafından temsil edilen tek aşamalı algılama modelleri nesne tanımlama ve algılama problemlerinde ağırlıklı olarak kullanılmaktadır. Hedef tespit modelinin ana performans göstergeleri tespit doğruluğu ve hızıdır. Doğruluk için hedef tespiti sadece sınıflandırma doğruluğunu değil, nesnenin konumlandırma doğruluğunu da dikkate alınmalıdır. Bu İki çeşit modellerin doğruluk, hassasıyet ve hesaplama verimliliği açısından performansları farklılıkları mevcuttur [30].

2.4.1. İki aşamalı algılama modelleri

2000'li yılların başında, makine donanımı deneysel gereksinimleri karşılamasiyla, derin sinir ağları hızla gelişti. CNN önerilmeden önce, insanlar genellikle hedef tespit problemi için geleneksel nesne tanima modellerini kullanıyorlardı. Evrişimsel sinir ağlarının kilometre taşı olan ResNet, GoogleNet ve AlexNet gibi bir dizi evrişimli sinir ağı modeli önerilmesiyle, görüntüdeki öznitelikler iyi bir şekilde çıkarılabilmeye başladı ve nesne tespitinin tespit doğruluğu büyük ölçüde geliştirildi. İki Aşamalı nesne algılama algoritması, önce bölgesel aday kutuları oluşturur, yani nesneleri içerebilecek önceden seçilmiş bir kutuyu bularak ve ardından evrişimli sinir ağları aracılığıyla sınıflandırma ve regresyon düzeltmesi yaparak nesne algılama yapmaktadır.

2.4.1.1. Bölgesel tabanlı evrişimsel sinir ağı (R-CNN)

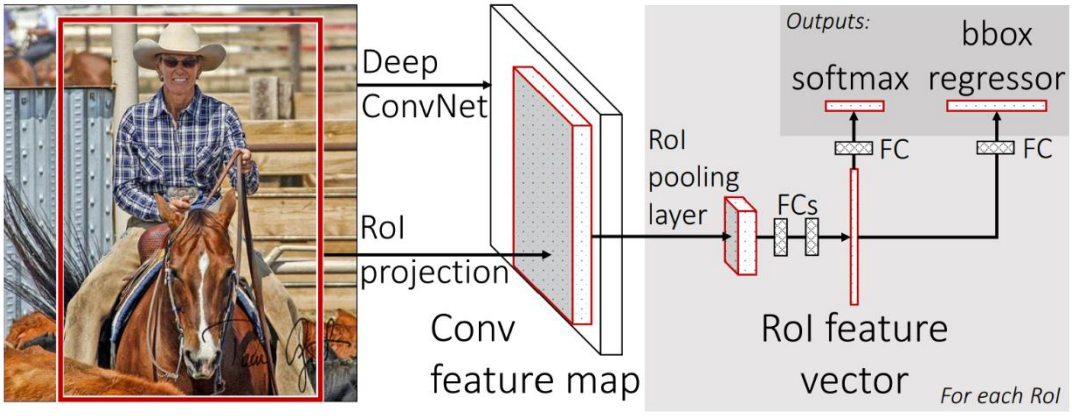
2014 yılında Girshick ve arkadaşları bölgesel tabanlı R-CNN'i önerdi. Bu algoritmada (Şekil 2.13.) nesnelerin bulunabileceği aday kutuları çıkarmak için aday bölge yöntemini kullanılır yanı görüntü 2000 kadar aday bölgelerine bölünür ve her bölge CNN ağından geçerek özellik çıkartması yapılmaktadır. Ardından çıkarılan her öznitelik SVM sınıflandırmasından geçirilir ve seçilen kutu için doğru konumu bulmak için regresör kullanılmaktadır. Son olarak, her bölgedeki nesnelerin sınıfları tahmin edilerek çıktı katmanından çıkarılmaktadır [31], [32].



Şekil 2.13. R-CNN'in sınıflandırma aşaması [32]

2.4.2. Hızlı R-CNN (Fast R-CNN)

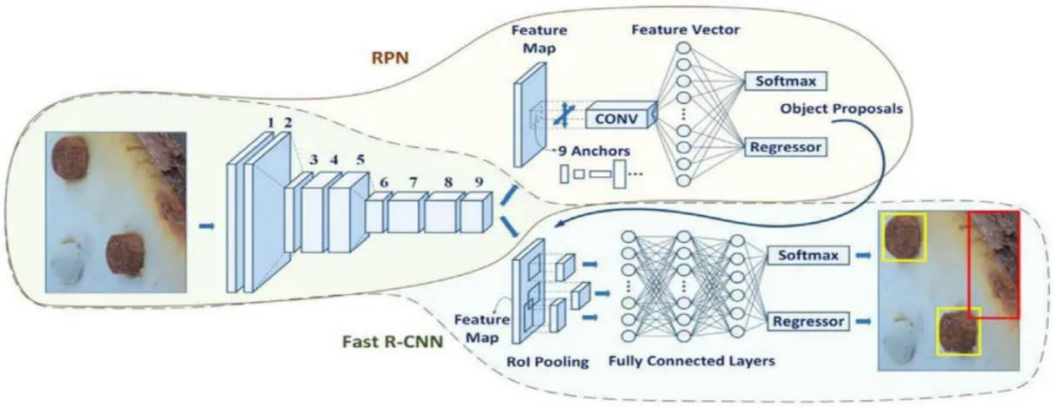
2015 yılında Girshick, R-CNN ve SPPNet'in daha da geliştirilmesi olan Fast RCNN nesne tespit algoritmasını önerdi. Fast R-CNN, R-CNN ve SPPNet'in avantajlarını bir araya getirerek algılayıcını ve sınırlayıcı kutu regresörünü aynı ağ yapılandırması altında aynı anda eğitmesını sağlamaktadır. Fast R-CNN ağı, tüm görüntü ve Bölge Teklifini (Regional Proposal) girdi olarak almaktadır (Şekil 2.14.). SPPNet'e benzer şekilde, özellik haritası evrişim katmanı tarafından çıkarıldıktan sonra, RoI havuzundan sonra sabit boyutlu özellik haritası çıkartılır ve ardından girdi olarak tamamen bağlı katman işlenir. Ancak SPPNet ve R-CNN'den farklı olarak, tam bağlı katmandan sonraki özellik vektörü aynı seviyedeki iki çıktı katmanına ayrılır: biri sınıflandırma için kullanılmaktadır, diğeri sınırlayıcı kutu regresyonu için kullanılır ve algılama kutusunun konum bilgisinin düzeltilmiş değerini verir [33] [34].



Şekil 2.14. Fast R-CNN mimarisi [33]

2.4.2.1. Daha hızlı R-CNN (Faster R-CNN)

R-CNN'nin daha hızlı hızlandırılması için Faster R-CNN geliştirildi. Fast R-CNN'nin Bölge Teklif (Regional Proposal) algoritması Seçici Arama (Selective Search) olduğundan, yalnızca CPU'da çalıştırılabiliyordu ve bu bir CNN ağı olmadığı için Fast R-CNN gerçek bir uçtan uca ağ değildi. Ren ve arkadaşları [35], Bölge Teklif (Regional Proposal) algoritmasını bir CNN ağı değiştirerek GPU altında çalışıp uçtan uca tam olarak çalışmasını sağlayan RPN ağını önermektedir. Şekil 2.15.'de gösterildiği gibi daha hızlı R-CNN'nin Seçici Arama algoritmasını (Selective Search Algorıthm) yerine aday kutuları oluşturmak için bir RPN ağı kullanılmaktadır. İlk olarak, girdi görüntüsünün bir özellik haritası elde etmek için CNN ağından geçirilir ve ardından aday kutuyu oluşturmak için RPN ağı kullanılır ve daha sonra aday kutusu özellik haritası alanında RoI havuzu ile sabit ölçekli bir özellik vektörü oluşturur. Sonra tamamen bağlı katmana girerek sınıflandırma ve çerçeve regresyonu işlemini tamamlar.



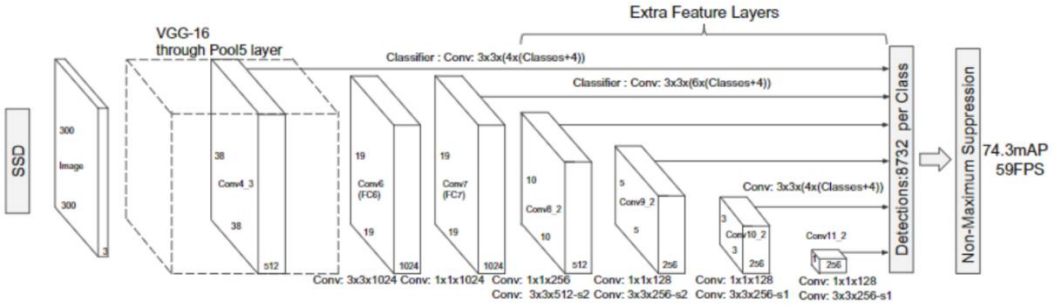
Şekil 2.15. Faster R_CNN'in çalışma adımları [35]

2.4.3. Tek aşamalı algılama modelleri

Tek Aşamalı Algılama algoritması ise bölge önerme aşaması gerektirmeyen ve doğrudan nesnenin kategori olasılığını ve konum koordinat değerini üreten yöntemdir. Tek bir algılamadan sonra nihai algılama sonucu doğrudan elde edilebilir. Öte yandan, tek aşamalı bir algılayıcı, sinir ağından yalnızca tek bir geçiş gerektirir ve tüm sınırlayıcı kutuları tek seferde tahmin eder. Bu, mobil cihazlar için çok daha hızlı ve çok daha uygundur. Tek aşamalı nesne algılayıcıların en yaygın örnekleri YOLO, SSD, SqueezeDet ve DetectNet'tir [36].

2.4.3.1. SSD

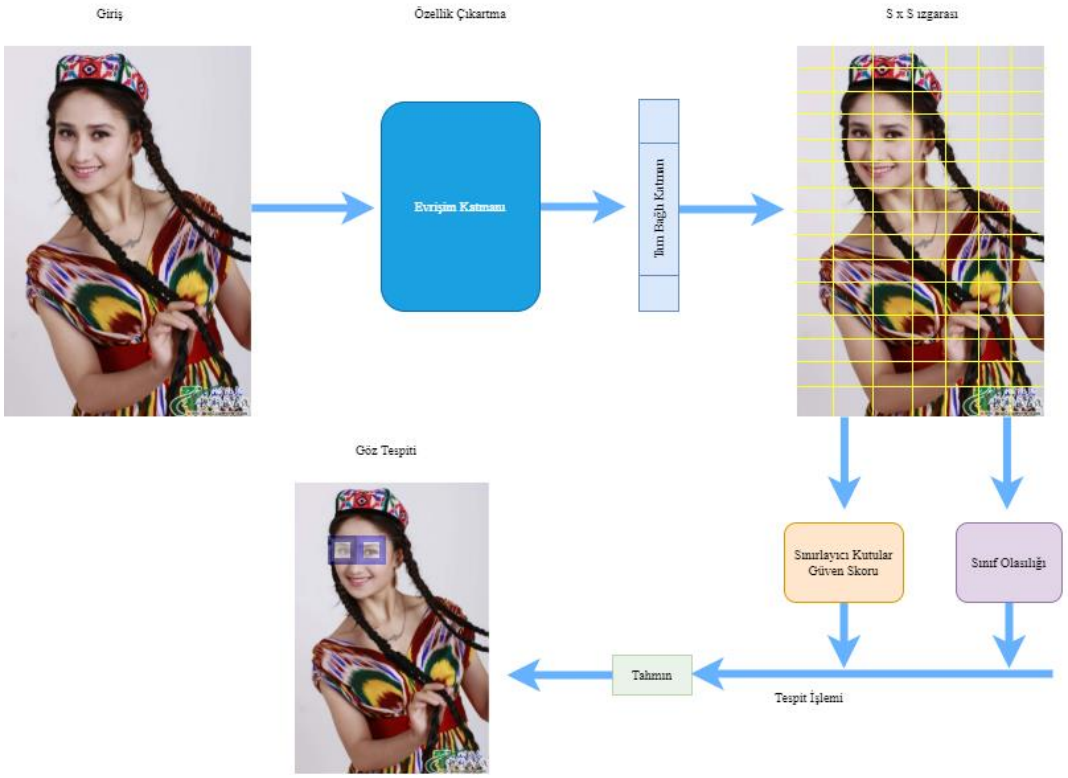
Derin öğrenme çağında ikinci tek aşamalı algılayıcı olan Tek-Seferde Çoklu Kutu Algılayıcı (Single Shot Detector, SSD), Liu ve diğerleri tarafından 2015 yılında önerilmiştir. SSD'nin ana katkısı, özellikle bazı küçük nesneler için tek aşamalı algılayıcıların algılama doğruluğunu büyük ölçüde artıran çoklu referans ve çoklu çözünürlüklü algılama tekniklerinin tanıtılmasıdır. SSD ile önceki herhangi bir algılayıcı arasındaki temel fark, birincisinin ağın farklı katmanlarında farklı ölçekteki nesneleri algılaması, ikincisi ise algılamayı yalnızca en üst katmanında çalıştırmasıdır. SSD'nin omurgası bir VGG'dir (Şekil 2.16.). Bu, Faster R-CNN ile aynıdır, ancak tam bağlı katmanı bir evrişim katmanıyla değiştirildiğinden, ön evrişim katmanı küçük nesneleri tahmin etmekten sorumludur ve sonraki evrişim katmanı ise büyük nesneleri tahmin etmekten sorumludur [37].



Şekil 2.16. SSD mimarisi [37]

2.4.3.2. YOLO

YOLO (You Only Look Once), gerçek zamanlı nesne algılayan tek aşamalı algılamanın öncü çalışmasıdır. R-CNN ve diğerleri gibi İki aşamalı algılama algoritmaları, algılama problemini iki aşamaya böler, önce aday bölgeler oluşturur (Regional Proposal) ve ardından aday bölgeleri sınıflandırır. YOLO ise bölge önerme aşaması gerektirmeyen ve doğrudan nesnenin kategori olasılığını ve konum koordinat değerini üreten tek aşamalı algılama algoritmasıdır. Şekil 2.17.'de gösterildiği gibi, YOLO, nesne algılama görevini doğrudan bir regresyon sorunu olarak ele alır ve görüntüyü bir CNN ağından geçirerek aday alanı ve algılama aşamalarını bir araya getirmektedir. Her görüntü ızgara bölümlerine ayrılarak, her bölümde hangi nesnelerin olduğunu ve nerede olduklarını tek bir algılamadan sonra nihai algılama sonucu doğrudan elde edilebilir [38].

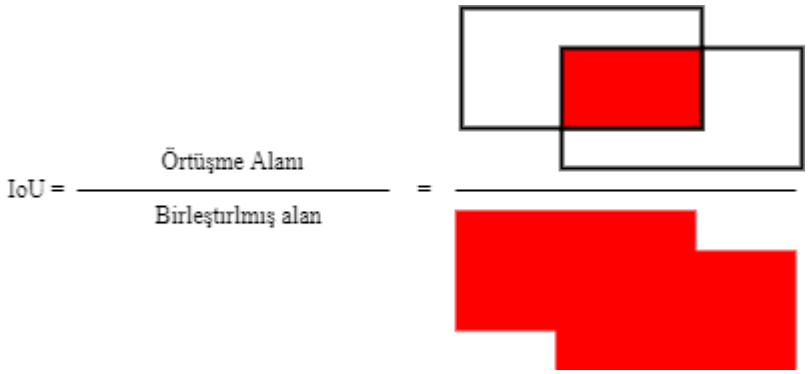


Şekil 2.17. YOLO'un göz tespitindeki çalışma adımları

Model görüntüyü SxS boyutlu ızgaralara böler ve her ızgaradaki görüntü özellikleri ile her bir hedef bölge ve kategoriyi tahmin eder. Eğer nesnenin merkezi ızgaraya düşerse ızgara nesneyi algılar. Her ızgara hücresi sınırlayıcı kutuyu hesaplar ve nesnelerin dahil edilip edilmediğini ve sınırlayıcı kutunun doğruluğunu değerlendiren güveni skorunu hesaplar. Her ızgara, B sınırlayıcı kutularını ve bu kutuya karşılık gelen güven değerini tahmin edecektir. Bu sınırlayıcı kutu, nesnenin merkez konumu (x, y), yüksekliği (h), genişliği (w) ve güven skorundan oluşan 5 parametreye sahiptir. Her kutu sadece B sınırlayıcı kutuları tahmin etmekle kalmaz, aynı zamanda kategorinin one-hot kodlama ile temsil edildiği kutudaki nesnenin kategorisini tahmin etmekten de sorumludur. Güven skoru, kutuda bir nesnenin bulunduğundan ne kadar güvenilir olduğunu ve kutunun doğruluğunu göstermektedir. Eğer bir ızgara hücresinde bir nesne yok ise onun güven skoru sıfırdır ve aksine bu hücrede nesnenin bulunduğu durumlarda ise güven skorunun tahmin edilen kutu ile temel gerçek (ground truth) arasındaki kesiştirilmiş bölgeler (IoU) ile örtüşmenin oranıdır (Denklem 2.1) [39] [40].

$$\text{Güven Skoru} = \Pr(Object) * IOU_{pred}^{truth} \tag{2.1}$$

IoU (Intersection over Union), Kesiştirilmiş Bölgeler olarak da bilinir, Şekil 2.18.'deki gibi birleştirilmiş sınırlama kutusu, tahmin edilen sınırlama kutusu ile gerçek sınırlama kutusu üzerinde ne kadar doğru konumlandırıldığını ölçen bir fonksiyondur [41].



Şekil 2.18. IoU hesaplama yöntemi

Hücre tarafından tahmin edilen sınırlayıcı kutunun her kategoriye ait olma olasılığını temsil eden C koşullu sınıfının tahminine "Pr ($Class_i$ | Object)" ilişkin bir olasılık değerini tahmin etmektedir. Her sınırlayıcı kutu için sınıfa özgü güven skorlarını (Denklem 2.2) 'deki gibi hesaplanır.

$$\Pr(Class_i \mid Object) * \Pr(Object) * IOU_{pred}^{truth} = \Pr(Class_i) * IOU_{pred}^{truth} \quad (2.2)$$

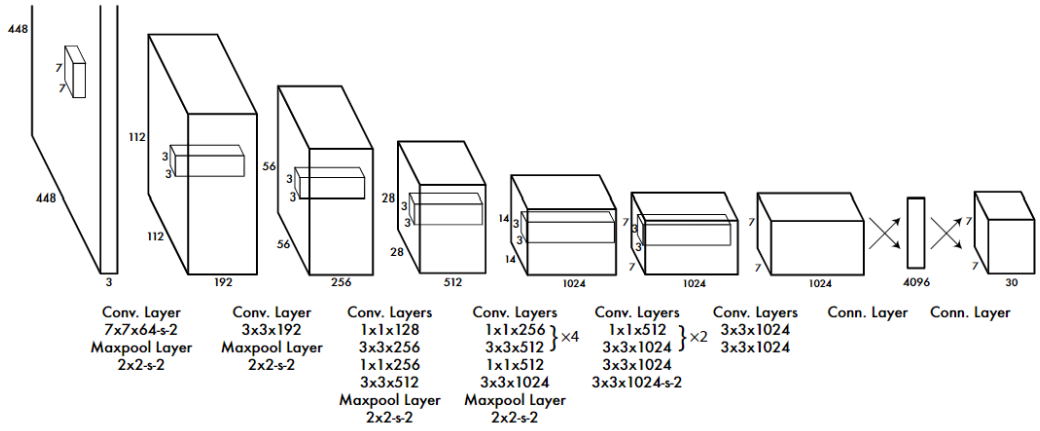
Bölünmüş ızgaradaki her hücre için birden fazla sınırlayıcı kutunun önerilmesi mümkündür. Model eğitilirken her bir sınırlayıcı kutu için bir nesnenin tahmin edilmesi beklenmektedir. Dolaysıyla gerçek doğruluğa (ground truth) en yakın IoU'ya sahip, güven kaybı en düşük olan bir tahminci atanmaktadır. Bu işlem sınırlayıcı kutu tahmin edicileri arasında en doğrusunu bulmak için yapılmaktadır. Tahmin edilen nesnenin, kutunun konumunun ve ızgara içinde nesne olup olmadığının ne kadar yanlış düzeyde olduğunu gösteren sınıflandırma, konum ve güven kaybından oluşan üç temel faktör YOLO'un kayıp fonksiyonunu oluşturmaktadır (Denklem 2.3).

$$\text{Kayıp Fonksiyon} = \lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{obj} [(x_i - \hat{x}_i)^2 + (y_i - \hat{y}_i)^2] +$$

$$\lambda_{\text{coord}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{obj}} \left[\left(\sqrt{w_i} - \sqrt{\hat{w}_i}\right)^2 + \left(\sqrt{h_i} - \sqrt{\hat{h}_i}\right)^2 \right] + \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{obj}} \quad (2.3)$$

$$(C_i - \hat{C}_i)^2 + \lambda_{\text{noobj}} \sum_{i=0}^{S^2} \sum_{j=0}^{B} I_{ij}^{\text{noobj}} (C_i - \hat{C}_i)^2 + \sum_{i=0}^{S^2} I_i^{\text{obj}} \sum_{c \in \text{classes}} (P_i(c) - \hat{p}_i(c))^2$$

YOLO, özellikleri çıkarmak için evrişimsel bir ağ kullanır ve ardından nesne olasılığı ve konum tahminlerini almak için tam bağlantılı bir katman kullanmaktadır. Ağ yapısı, Şekil 2.19.'da gösterildiği gibi 24 evrişimsel katman ve 2 tam bağlantılı katman içeren GooLeNet bir modelidir. Evrişimli katmanlar ve tam bağlantılı katmanlar için Leaky ReLU aktivasyon fonksiyonu kullanılmaktadır. Ancak son katman, doğrusal bir aktivasyon fonksiyonu kullanılmaktadır.



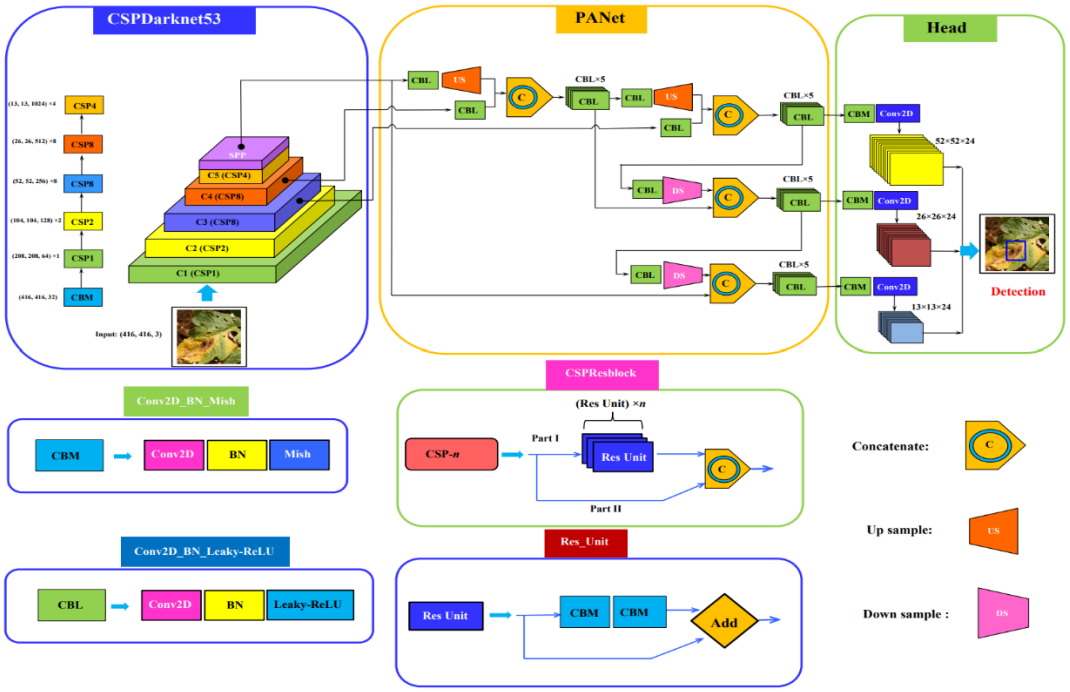
Şekil 2.19. YOLO'un nesne tespitinde kullanılan katmanlar [42]

2.5. YOLOv4'un Mimarisi

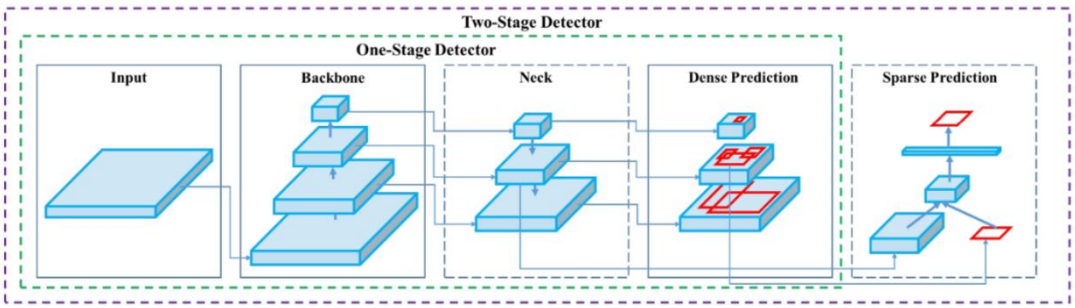
Alexey ve arkadaşları YOLO serisinin fikir ve kavramlarını devraldı ve YOLOv3 temelinde sürekli olarak iyileştirildi, geliştirildi ve Nisan 2020'de YOLOv4'ü piyasaya sürdü. YOLOv4 algoritması, son yıllarda CNN alanındaki en iyi optimizasyon stratejisini benimsemektedir. Veri işleme, omurga ağı, ağ eğitimi, aktivasyon fonksiyonu, kayıp fonksiyonu gibi farklı alanlarda daha önceki modellere göre daha optimizasyon sergilemektedir. YOLOv4, geleneksel GPU'lar kullanılarak eğitilebilir ve test edilebilir ve gerçek zamanlı, yüksek hassasiyetli algılama sonuçları elde

etmiştir. Diğer son teknoloji nesne dedektörleri ile karşılaştırıldığında performans ve çıkarım hızında önemli artış göstermiştir [43].

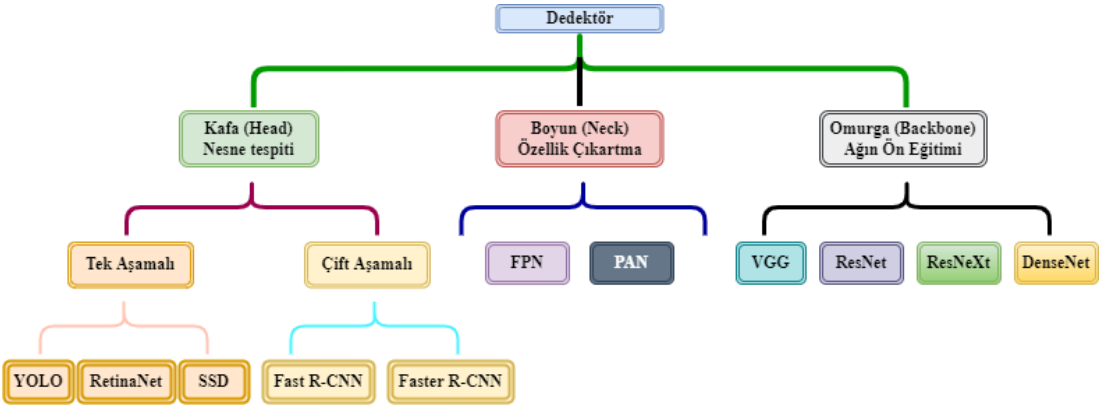
YOLOv4'ün genel ağ mimarisi (Şekil 2.20.), Omurga (Backbone), Boyun (Neck) ve Kafa (Head) kısımlarından oluşmaktadır (Şekil 2.21.). Omurga, farklı görüntülerden detaylı bir düzeyde görüntü özelliklerini toplayan ve oluşturan evrişimli bir sinir ağıdır. Boyun ise görüntü özelliklerini işleyen ve onu tahmin katmanına ileten bir dizi ağ katmanıdır. Kafa kısmi ise görüntü özellikleri üzerinde tahminler yapan, sınırlayıcı kutular oluşturan ve kategorileri tahmin eden katmandır (Şekil 2.22.) [43].



Şekil 2.20. YOLOv4 ağ mimarisinin şeması [44]



Şekil 2.21. Nesne tespit dedektörü [43]



Şekil 2.22. Nesne dedektörün yapısı

2.5.1. Omurga (Backbone)

YOLOv4, giriş görüntülerinden zengin bilgilendirici özellikler çıkarmak için Omurga olarak CSPDarknet'i kullanır. CSPNet [44], diğer büyük ölçekli evrişimli sinir ağı çerçevesi Omurgasındaki ağ optimizasyonunun gradyan bilgisi tekrarlama problemini çözmektedir ve gradyan değişikliğini baştan sona özellik haritasına entegre etmektedir, böylece modelin parametre miktarını ve FLOPS değerini azaltmaktadır ve modelin boyutunu küçülterek çıkarım hızını ve doğruluğunu sağlamaktadır. CSPNet aslında Densnet [45]'in temel katmanın özellik haritasının kopyalanması ve kopyanın yoğun blok üzerinden bir sonraki aşamaya gönderilmesi, böylece temel katmanın özellik haritasının ayrılması fikrine dayanmaktadır. Bu, gradyan kaybı problemini etkili bir şekilde hafifletebilir, özellik yayılımını destekliyor ve ağı özellikleri yeniden kullanmaya teşvik ederek ağ parametrelerinin sayısını azaltmaktadır ve böylece ağın doğruluğunu artırmaktadır [43], [44], [46]. YOLOv4 tarafından kullanılan teknikler ve yapılan iyileştirmeler Tablo 2.1.'de gösterilmektedir.

Tablo 2.1. YOLOv4'te kullanılan teknikler

Geliştirilmiş Kısımlar	Kullanılan Teknikler
Activation	Mish [47]
Bounding box regression loss	GIoU/CIoU [48]
Data augmentation	Mosaic Self-Adversarial Training (SAT)
Regularization method	DropBlock [49]
Normalization	Improved Cross mini-Batch Normalization [50]

YOLOv4'ün Omurga kısmı Bag-of-Freebie ve Bag-Of-Specials gibi yenilikleri içermektedir. Bag-of-Freebie, çıkarım hızı üzerinde hiçbir etkisi olmayan ve

doğruluğu artıran eğitim sürecinde yapılan veri büyütme, sınıf dengesizliği, maliyet işlevi, yumuşak etiketleme gibi özel iyileştirmeleri içerir. Bag-Of-Specials, Ağı iyileştirerek, iyi bir performans getirisi ile çıkarım süresini biraz etkiler. Bu iyileştirmeler, alıcı alanın artırılmasını, dikkatin kullanılmasını, özellik entegrasyonunu ve maksimum olmayan bastırma gibi son işlemeyi içerir [51].

2.5.2. Boyun (Neck)

Boyunun farklı boyutlardaki nesneleri algılamak için, kafanın farklı uzaysal çözünürlüklerdeki özellik haritalarını algılayabilmesi için hiyerarşik bir yapının kullanılması gerekir. Kafayı (Head) besleyen bilgiyi zenginleştirmek için, aşağıdan yukarıya akıştan ve yukarıdan aşağıya akıştan gelen komşu özellik haritaları, kafaya beslenmeden önce eleman bazında birbirine eklenir veya birleştirilir. Bu nedenle, kafanın girdisi aşağıdan yukarıya akıştan uzaysal zengin bilgiyi ve yukarıdan aşağıya akıştan anlamsal zengin bilgiyi içermektedir. Boyun esas olarak özellik piramitleri oluşturmak için kullanılır. Özellik piramidi, modelin farklı boyut ve ölçeklerdeki aynı nesneyi tanıyabilmesi için farklı ölçeklerdeki nesneleri algılamasını geliştirir. PANET [52]'in ortaya çıkışından önce, FPN [53], PANET'in ortaya çıkışına kadar nesne algılama çerçevesinin özellik toplama katmanının en son teknolojisi olmuştur [54].

2.5.3. Baş (Head)

YOLOv4'in baş kısmı nesneleri tespit etmek ve onların konumlarını belirlemek için YOLOv3[55] kullanmaktadır. Ağ, bir sınıf için sınırlayıcı kutu koordinatlarını (x, y, w, h) ve güven puanını algılar. YOLO'nun amacı, görüntüyü birden fazla hücreden oluşan bir ızgaraya bölmek ve ardından her hücre için öneri kutularını kullanarak bir nesneye sahip olma olasılığını tahmin etmektir. Çıktı, sınırlayıcı kutu koordinatlarına ve olasılık sınıflarına sahip bir vektördür.

BÖLÜM 3. DERİN ÖĞRENME İLE İNSAN GÖZÜNÜN TESPİTİ

Bu bölümde yapay zekâ teknikleri kullanarak haraketli görüntülerden insan gözünün tespit edilmesi çalışmasındaki gerekli materyal ve yazılımlarla ilgi çalışmalar sunulmaktadır. Göz tespiti için Bölüm 2'de anlatılan YOLOv4 kullanılmaktadır ve bu tespitin yapılması için gereken veri setleri, veri hazırlama yazılımları ve yazılım ortamları ele alınmaktadır.

3.1. Yazılım

Bu çalışmada görüntülerden göz tespiti yapabilmek için farklı derin öğrenme kütüphaneleri kullanılmıştır. Göz tespiti modelleri eğitmek için Numpy, Darknet gibi kütüphaneler kullanılmıştır. Bunların dışında verilerin boyutlandırma işlemlerini yapmak için OpenCV, YOLOv4 modelini eğitmek için kullanılan veri setlerini etiketleme işleri için web tabanlı çalışan MakeSensise kullanılmiştir.

3.1.1. Makesense

Makesense.ai[56], fotoğrafları etiketlemek için kullanımı ücretsiz bir çevrimiçi araçtır. MakeSense[57], GPLv3 lisansı altında açık kaynaklı ve kullanımı ücretsiz bir görüntü etiketleme aracıdır. Herhangi bir kurulum gerektirmez, web tarayıcı kullanımı sayesinde ihtiyaç duyduğumuz etiketleme işlemlerini hızla yapabildiğimiz araçtır. Kullanıcı ara yüzü basit ve kullanımı kolaydır. Etiketlemek istediğiniz görüntüleri MakeSense'e yükleyip, görüntülere etiketlememiz ve daha sonra etiketleri YOLO, VOC XML, VGG JSON ve CSV dahil olmak üzere farklı formatlarda aktarma imkanı sağlamaktadır.

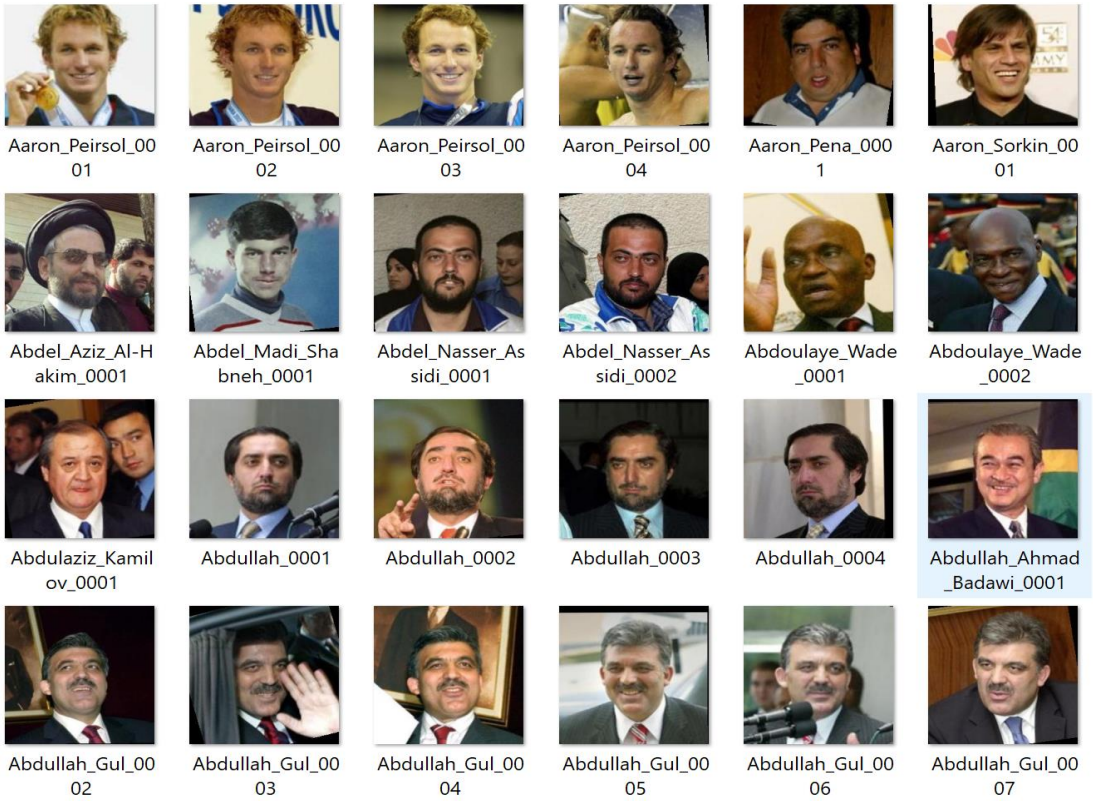
3.1.2. OpenCV

Açık Kaynaklı Bilgisayarla Görme Kütüphanesi (OpenCV) [58], 1999 civarında bir Intel araştırma girişimi olarak başlamıştır. Şimdi, bilgisayarla görme ve makine öğrenimi için en popüler açık kaynaklı yazılım kitaplığıdır. Başlangıçta, görüntü işleme ve bilgisayarla görme için bir dizi C kütüphanesi fonksiyonudu. Artık C++, Python, Java ve MATLAB destekleri var ve CUDA [59] ve OpenCL'den hızlandırma desteğiyle macOS, Windows, Linux, Android ve iOS üzerinde çalışmaktadır. OpenCV kütüphanesi bir modül koleksiyonu olarak gelimiştir. Modüllerin her biri, görüntü işleme, bilgisayarla görme ve makine öğrenimi çatısı altında belirli bir uygulama grubunu yönetir. OpenCV, hesaplama verimliliği için ve gerçek zamanlı uygulamalara güçlü bir şekilde odaklanarak tasarlanmıştır. Optimize edilmiş C++ ile yazılmıştır ve çok çekirdekli işlemcilerden yararlanabilir [60], [61].

3.2. Veri Seti

Bu çalışmada farklı iki yüz tanıma veri setlerindeki görüntülerden veri seti oluşturulmuştur. Labeled Faces in the Wild (LFW)[62] olarak adlandırılan yüz tanıma problemleri için tasarlanmış yüz görüntülerinden oluşan bu veri seti, internet üzerinden toplan 13.000'den fazla yüz görüntüsünü içerir ve her resim, ona ait olan kişinin adıyla etiketlenmiş. Görüntülerdeki 1680’inin iki veya daha fazla farklı resimleri mevcuttur ve resimlerin boyutları 250x250’dir. Face Detection Data Set and Benchmark (FDDB)[6]’da bir başka yüz algılama veri setidir ve 2845 görüntüden oluşmaktadır ve görüntülerin boyutları ise farklı boyutlarda mevcuttur. Çalışmada toplam 3009 görüntü kullanılmiştir ve bu görüntünün 1009’u FDDB veri setinden (Şekil 3.1.), 2000’i LFW veri setinden alınmiştır (Şekil 3.2.).

Şekil 3.1. FDDB veri setindeki görüntüler

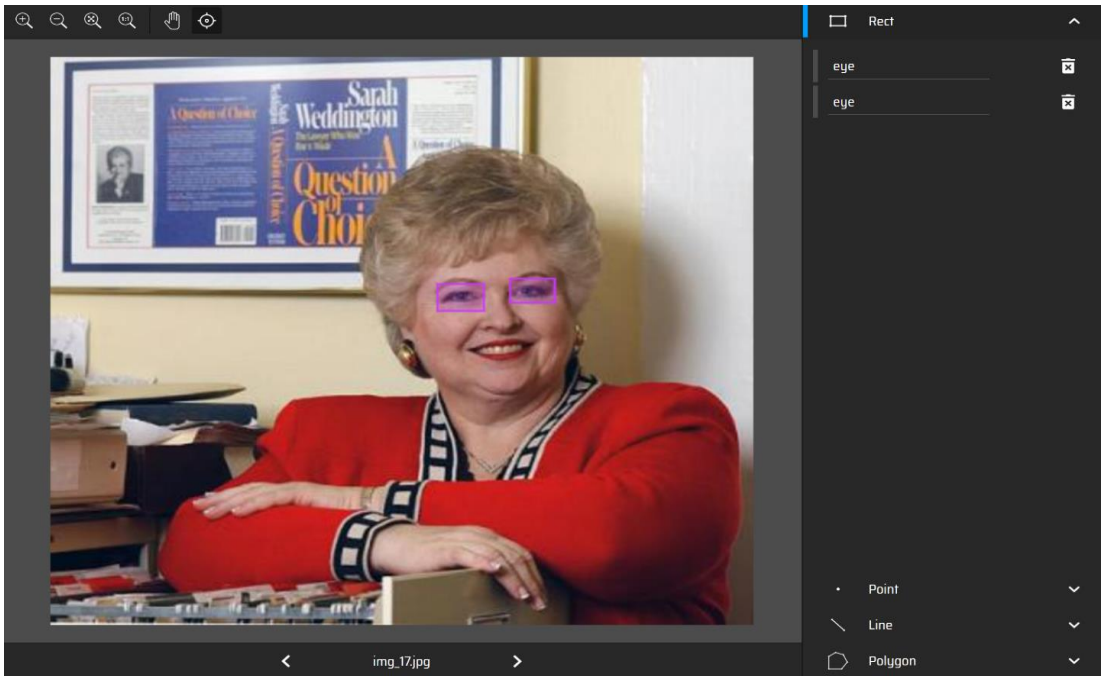


Şekil 3.2. LFW veri setindeki görüntüler

3.3. Veri Setinin Hazırlanması

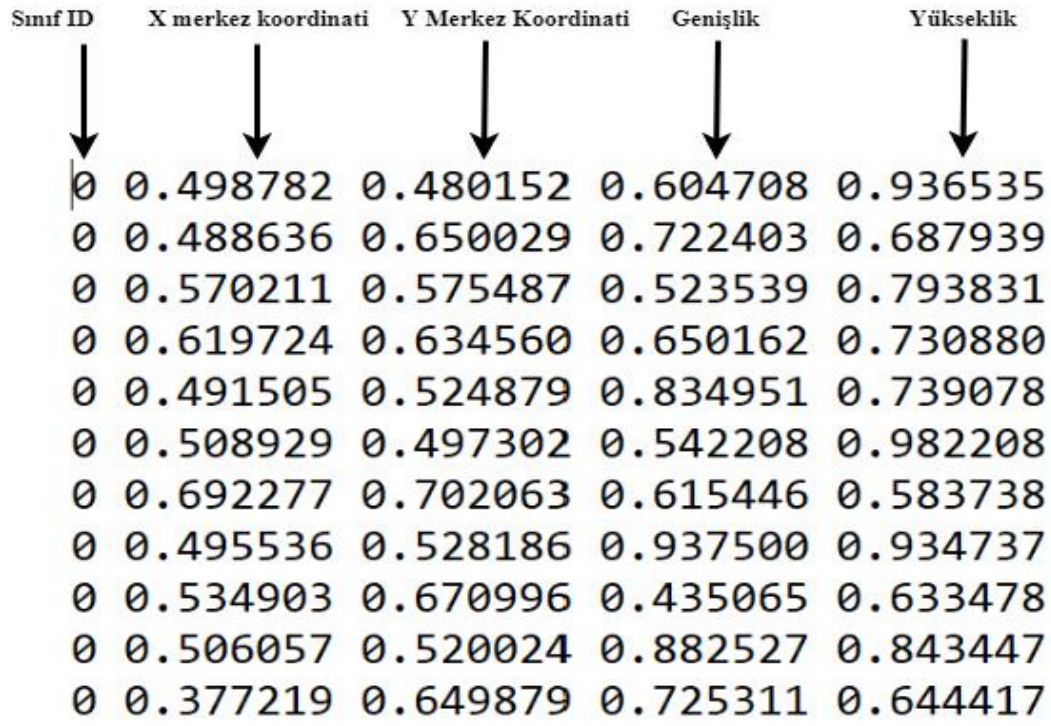
Veri setindeki görüntüler farklı boyutlarda mevcuttur ve YOLOv4 modeli için kullanacağımız görüntülerin boyutları 416x416 olduğundan veri setimizdeki

görüntüleri bu boyutlara ayarlamamız gerekmektedir. Veri setindeki görüntülerin boyutlarını 416x416'ye ayarlamak için OpenCV kütüphanesinin resize fonksiyonu kullanılmiştir ve böylelikle 3009 görüntünün boyutları YOLOv4 model için gerekli olan boyuta ayarlanmıştır. Daha sonra nesne tanıma işlemlerinde en çok zaman alan fotoğraf etiketleme işlemi web tabanlı, kolay fotoğraf etiketleme aracı olan Makesense ile yapılmıştır. Görüntüler Makesense'e yükledikten sonra her bir fotoğraf için ayrı ayrı etiketleme işlemi yapmamız gerekmektedir. Şekil 3.3.'te Makesense'ın görüntü etiketleme ara yüzünü göstermektedir.

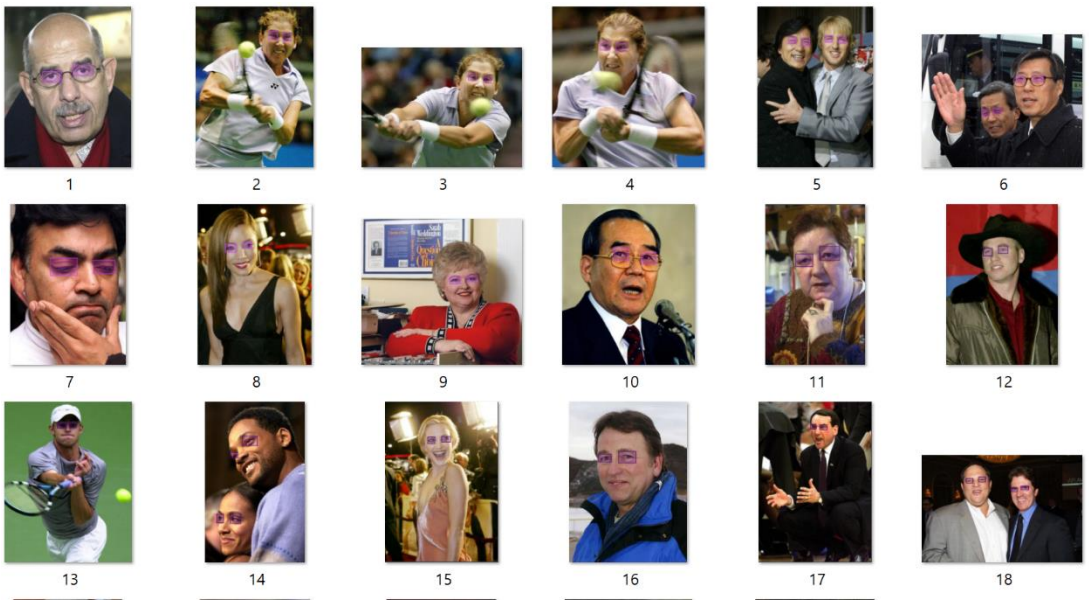


Şekil 3.3. Makesense'in etiketleme ara yüzü

Şekil 3.5.'te Makesense ile etiketlenmiş fotoğraflar gösterilmektedir. Etiketlenme işlemleri tamamlandıktan sonra her fotoğraflar için Şekil 3.7.'deki gibi bir etiket dosyası oluşturmaktadır. Şekil 3.8'de gösterilen etiket dosyası beş parametreden oluşmaktadır ve Şekil 3.4.'te gösterildiği gibi ilk parametre etiketin sınıfını göstermektedir ve geri kalan 4 parametre ise 0 ile 1 arasında normalize edilmiş olup bu sınıfın sınırlayıcı kutusunun fotoğraftaki x ve y merkezi konum koordinatları ve etiketin genişliği ve yüksekliğini içermektedir. Şekil 3.6.'te etiketlenmemiş normal fotoğrafları gösterilmektedir.

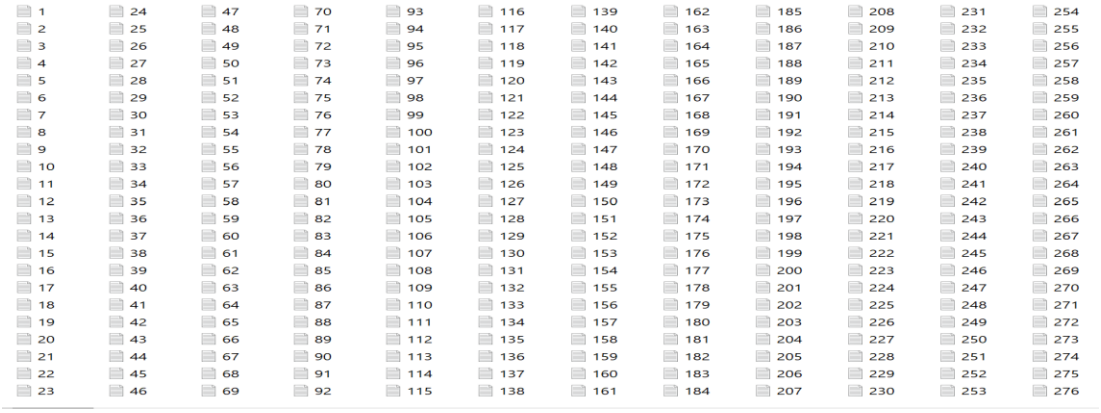


Şekil 3.4. Etiket parametrelerinin açıklaması



Şekil 3.5. Gözleri etiketlenmiş fotoğraflar

Şekil 3.6. Etiketlenmemiş fotoğraflar



Şekil 3.7. Etiket dosyası

```
0 0.498782 0.480152 0.604708 0.936535
0 0.488636 0.650029 0.722403 0.687939
0 0.570211 0.575487 0.523539 0.793831
0 0.619724 0.634560 0.650162 0.730880
0 0.491505 0.524879 0.834951 0.739078
0 0.508929 0.497302 0.542208 0.982208
0 0.692277 0.702063 0.615446 0.583738
0 0.495536 0.528186 0.937500 0.934737
0 0.534903 0.670996 0.435065 0.633478
0 0.506057 0.520024 0.882527 0.843447
0 0.377219 0.649879 0.725311 0.644417
0 0.566558 0.510823 0.581169 0.974026
0 0.418425 0.537120 0.796266 0.697378
0 0.534587 0.638350 0.682646 0.703883
0 0.449901 0.646238 0.663288 0.508495
0 0.471481 0.503034 0.714806 0.690534
0 0.446023 0.704682 0.325487 0.590636
0 0.171266 0.486005 0.259740 0.230588
0 0.795860 0.593301 0.249188 0.173274
0 0.958604 0.627956 0.082792 0.149282
0 0.551136 0.660534 0.378247 0.678932
0 0.506878 0.502427 0.855603 0.978155
0 0.608766 0.599928 0.639610 0.800144
```

Şekil 3.8. Görüntü etiketlendikten sonra oluşan etiket parametreleri

3.4. Veri Setinin Ayırılması

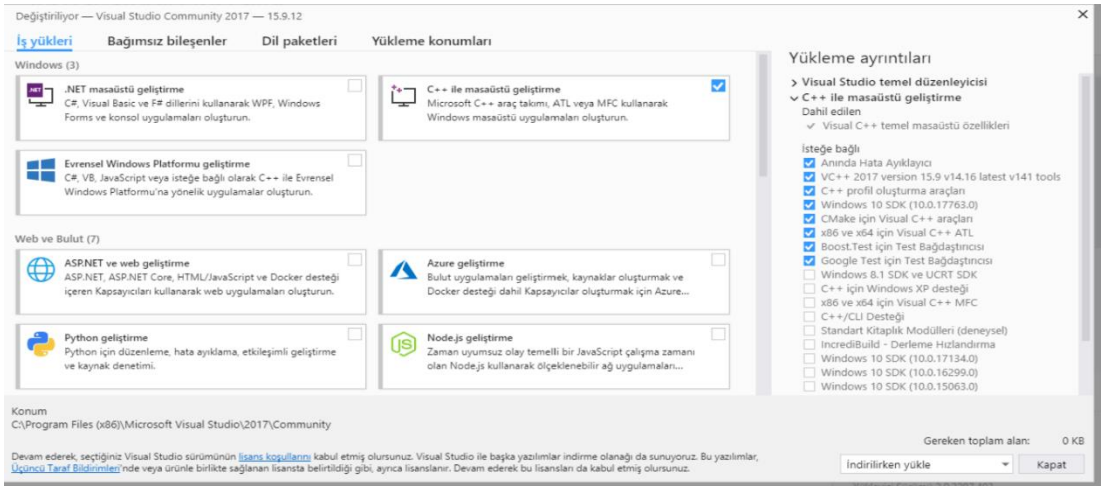
Derin öğrenme problemlerinde modellerin eğitilmesi için gereken veri setleri eğitim, test ve doğrulama gibi setlere farklı oranlarda ayırılmaktadır. Verilerin farklı setlere ayırarak modelin eğitim, test ve doğrulama aşamalarında kullanılması modelin iyi eğitilmesi ve iyi performans sağlamasını amaçlamaktadır. Bu çalışmada kullanılacak görüntü verileri eğitim ve test seti olmak üzere iki farklı sete ayırılmaktadır. Görüntü verilerimiz daha önceki bölümlerde bahsettiğimiz iki farklı görüntü setinden alınan 3009 görüntüden oluşmaktadır. Bu görüntülerin %80'lık bölümü yanı 2407 adet görüntü eğitim seti ve %20'lık bölümünden oluşan 602 adet görüntü test seti olarak ayırılmıştır. Makesense ile yapılan veri etiketleme işleminden sonra elde ettiğimiz YOLO formatındaki 3009 etiket de eğitim ve test setindeki görüntüler ile setlere ayırılmıştır.

3.5. Göz Koruma Kontrolü Modeli İçin Veri Seti

Veriseti internet üzerinden toplanmış 1100 görüntüden oluşmaktadır. Görüntülerden 523'ü koruyucu gözlük kullananlara aittir ve geri kalan 578 görüntü normal gözlük takan ve takmayanlardan oluşmaktadır. Görüntüler modele verilmeden önce OpenCV ile YOLOv4 modeline uygun şekilde boyutlandırılmıştır. Veri setinin %80'i eğitim ve %20'lık bölmü test seti için ayırılmiştir. Daha sonra web tabanlı görüntü etiketleme yazılımı Makesense ile görüntülerin etiketleme işlemleri yapılmıştır. Görüntüler koruyucu gözlüklü ve koruyucu gözlüksüz şeklinde iki sınıf ile etiketlenmiştir.

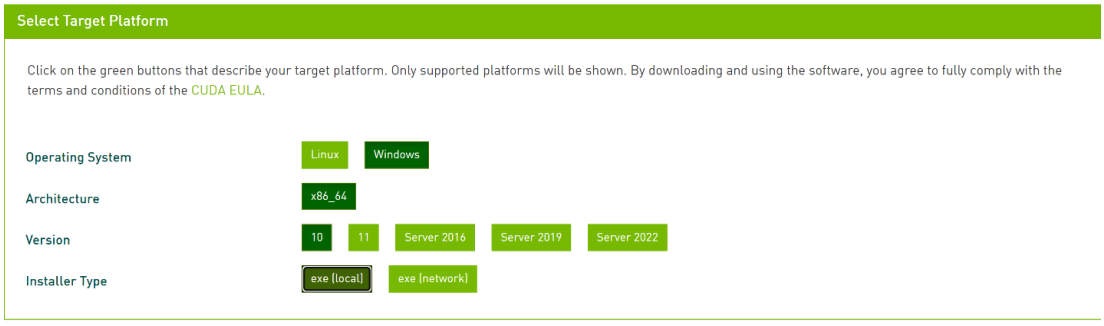
3.6. Yazılım Ortamının Konfigürasyonu

YOLOv4 kendi bilgisayarımızda çalıştırabilmek için bazı yazılımların yüklenmesi ve konfigürasyon işlemlerinin yapılması gerekmektedir. İlk olarak Visual Studio Community [63] sürümünün yüklenmesi gerekmektedir ve bunu yüklerken "C++ ile masaüstü geliştirme" paketinin yüklenmesi yeterli olacaktır (Şekil 3.9.).

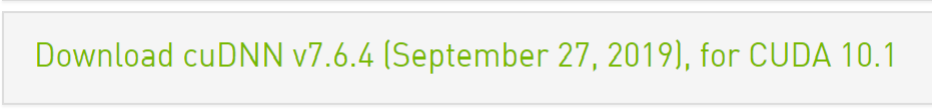


Şekil 3.9. Visual Stüdyo'nu gerekli paketinin yüklenmesi

YOLOv4 bilgisayarın GPU'su üzerinde çalışması için NVİDİA [59]'in bilgisayarımızdaki donanımlara uygun CUDA [64] GPU ve cuDNN [65] paketlerini yüklenmesi gerekmektedir. Bu paketlerden bilgisayardaki GPU'a uyun olan sürümünü Şekil 3.12.'de bulabiliriz. CUDA GPU paketini yükledikten sonra cuDNN paketini de yüklenmesi gerekmektedir. Bu çalışmada bilgisayardaki GPU donanımına uygun sürümü olarak CUDA'in 10.sürümü ve cuDNN ise 7.sürümü yüklenmiştir (Şekil 3.10. ve Şekil 3.11.).



Şekil 3.10. CUDA 10 sürümü [64]



Şekil 3.11. cuDNN 7.6.4 sürümü [65]

Modelin çalışmasında önemli olan OpenCV, Python [66], Numpy [67] ve CMake [68] gibi yazılımlarda yüklenmiştir. Bu yazılımları sürümleri ile bilgiler Tablo 3.1.'de

gösterilmektedir. Bu yazılımların yüklenmesi tamamlandıktan sonra YOLOv4 modelimizi çalıştırabilmek için Darknet [69]'i yüklememiz gerekmektedir. Darknet, C ve CUDA ile yazılmış açık kaynaklı Evrişimli Sinir Ağları biçiminde Nesne Algılama ve Görüntü Sınıflandırma görevlerini destekleyen bir çerçevedir. Hızlıdır, kurulumu kolaydır ve CPU ve GPU hesaplamasını destekler. Darknet, C ve CUDA'da yazılmış böyle bir açık kaynaklı sinir ağı çerçevesidir ve YOLO'nun temeli olarak hizmet eder. Hızlıdır, kurulumu kolaydır ve CPU ve GPU hesaplamasını destekler. Darknet, YOLO eğitimi için çerçeve olarak kullanılır, yani ağın mimarisini belirler.

Tablo 3.1. Yüklenmesi gereken yazılımların sürümü

Yazılım veya Paket	Sürümü
Visual Studio	Community 2019 16.5
CUDA	10.1
cuDNN	7.6.4
OpenCV	4.5.5
Python	3.8
CMake	3.23.0
Numpy	1.22.0

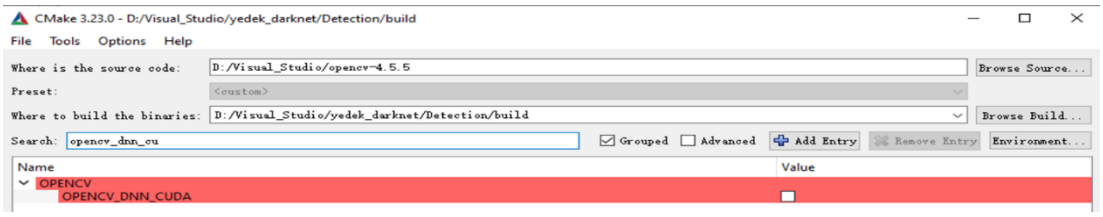
5.0	Maxwell	GM107, GM108	GeForce GTX 750 Ti, GeForce GTX 750, GeForce GTX 960M, GeForce GTX 950M, GeForce 940M, GeForce 930M, GeForce GTX 860M, GeForce GTX 850M, GeForce 845M, GeForce 840M, GeForce 830M	Quadro K1200, Quadro K2200, Quadro K620, Quadro M2000M, Quadro M1000M, Quadro M600M, Quadro K620M, NVS 810	Tesla M10	
5.2		GM200, GM204, GM206	GeForce GTX Titan X, GeForce GTX 980 Ti, GeForce GTX 980, GeForce GTX 970, GeForce GTX 960, GeForce GTX 950, GeForce GTX 750 SE, GeForce GTX 980M, GeForce GTX 970M, GeForce GTX 965M	Quadro M6000 24GB, Quadro M6000, Quadro M5000, Quadro M4000, Quadro M2000, Quadro M5500, Quadro M5000M, Quadro M4000M, Quadro M3000M	Tesla M4, Tesla M40, Tesla M6, Tesla M60	
5.3		GM20B				Tegra X1, Jetson TX1, Jetson Nano, DRIVE CX, DRIVE PX
6.0	Pascal	GP100		Quadro GP100	Tesla P100	
6.1		GP102, GP104, GP106, GP107, GP108	Nvidia TITAN Xp, Titan X, GeForce GTX 1080 Ti, GTX 1080, GTX 1070 Ti, GTX 1070, GTX 1060, GTX 1050 Ti, GTX 1050, GT 1030, GT 1010, MX350, MX330, MX250, MX230, MX150, MX130, MX110	Quadro P6000, Quadro P5000, Quadro P4000, Quadro P2200, Quadro P2000, Quadro P1000, Quadro P400, Quadro P500, Quadro P520, Quadro P600, Quadro P5000(Mobile), Quadro P4000(Mobile), Quadro P3000(Mobile)	Tesla P40, Tesla P6, Tesla P4	
6.2		GP10B[41]				Tegra X2, Jetson TX2, DRIVE PX 2
7.0	Volta	GV100	NVIDIA TITAN V	Quadro GV100	Tesla V100, Tesla V100S	
7.2		GV10B[42] GV11B[43][44]				Tegra Xavier, Jetson Xavier NX, Jetson AGX Xavier, DRIVE AGX Xavier, DRIVE AGX Pegasus, Clara AGX
7.5	Turing	TU102, TU104, TU106, TU116, TU117	NVIDIA TITAN RTX, GeForce RTX 2080 Ti, RTX 2080 Super, RTX 2080, RTX 2070 Super, RTX 2070, RTX 2060 Super, RTX 2060 12GB, RTX 2060,	Quadro RTX 8000, Quadro RTX 6000, Quadro RTX 5000, Quadro RTX 4000, T1000, T600, T400 T1200(mobile), T600(mobile), T500(mobile)	Tesla T4	

Şekil 3.12. GPU donanımına uygun yazılım sürümünün listesi [70]

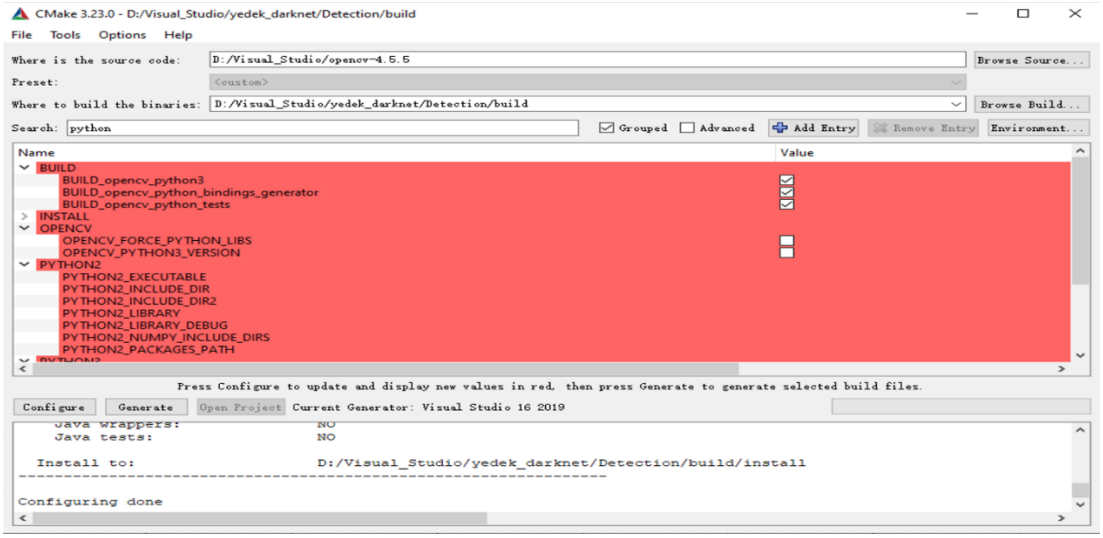
CMake, derleme sürecini bir işletim sisteminde ve derleyiciden bağımsız bir şekilde yöneten genişletilebilir, açık kaynaklı bir sistemdir. Birçok platformlar arası sistemin aksine, CMake yerel yapı ortamıyla birlikte kullanılmak üzere tasarlanmıştır. Her kaynak dizine yerleştirilen basit yapılandırma dosyaları, olağan şekilde kullanılan standart derleme dosyalarını oluşturmak için kullanılır. CMake, kaynak kodunu

derleyecek, kitaplıklar oluşturacak ve kombinasyonlar halinde yürütülebilir dosyalar oluşturacak yerel bir oluşturma ortamı oluşturabilir. CMake, çeşitli kitaplıklara bağlı karmaşık dizin hiyerarşilerini ve uygulamaları desteklemek için tasarlanmıştır. CMake, her bir araç setinin birkaç dizin içerebileceği ve uygulamanın araç setlerine ve ek koda bağlı olduğu birden çok araç setinden oluşan projeleri destekler [71].

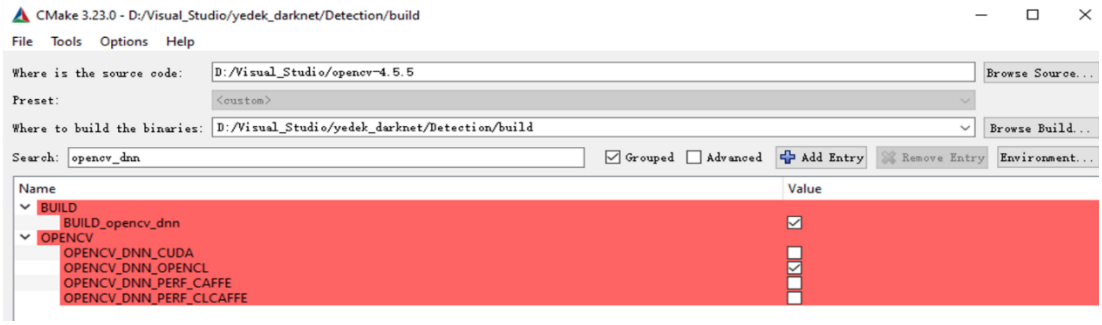
Modelimize temiz bir derleme ortamının oluşturabilmek için CMake ile Python, OpenCV ve CUDA'ların içinden derlemeniz gereken bağımlılıkların, çalıştırılabilir dosyaları, DLL'leri ve benzerlerini dosyaların konfigürasyonunu yapılmaktadır. Şekil 3.13., 3.14. ve 3.15.'lerde gösterilen Python, CUDA ve cuDNN seçenekleri işaretlenerek konfigürasiyonu yapılacaktır.



Şekil 3.13. cuDNN konfigürasyonu



Şekil 3.14. Python konfigürasyonu



Şekil 3.15. CUDA konfigürasyonu

Python, OpenCV ve CUDA'larla ilgili konfigürasyon tamamlandıktan sonra Darknet'te de bazı ayarlar yapmamız gerekmektedir. Modelimize ait olan "Yolov4-obj.cfg" dosyasında bazı değişikler Şekil 3.16.'de gösterildiği gibi yapılmaktadır. Bu dosyada 3 adet yolo parametresi mevcuttur bunu Şekil 3.17.'de gösterildiği gibi yapılması gerekmektedır. Bu dosyadaki aşağıdaki belirtilen parametrelerde değişiklik yapmamız gerekmektedir;

- Bach_size: Eğitim sırasında modele verilen görüntülerin parçalar halinde verilmesini sağlayan parametredir ve bunu 64 olarak ayarlayacağız.
- Subdivision: 64, Partimizi böldüğümüz mini partinin sayısını gösterir. Eğer kullandığımız GPU'lar güçlü hesaplama gücüne sahip ise ya da daha iyi belleğe sahip ise 4, 8, 16, 32 gibi daha düşük ayarları yapabiliriz.
- Width: 416
- Height: 416
- Max_batches: 4000, bu parametreyi ayarlarken Denklem 3.1'i referans alarak ayarlayabiliriz ama YOLOv4'ün yazarına Alexey [72] göre bu sayı eğitilecek görüntü sayısından da az olmamalı dediği için bu sayı modelimizde eğitilecek görüntü sayısının üzerinde ayarlanmaktadır.

$$Max\_batches = [sınıf\_sayısı * 2000] \quad (3.1)$$

- Steps: 3200, 3600. Bu parametreler Denklem 3.2 ve Denklem 3.3'te gösterildiği gibi hesaplanabilir.

$$Steps = [Max\_batches * \%80] \quad (3.2)$$

$$Steps = [Max\_batches * \%90] \quad (3.3)$$

– Filters: 18, bu Denklem 3.4'te gösterildiği gibi hesaplanmaktadır.

$$Filters = [Sınıf\_Sayısı \ + 5\ ] * 3 \quad (3.4)$$

– Classes: 1, Modelimizde bulunan sınıf sayısını göstermektedir. Bizim modelde sadece bir sınıf bulunmaktadır.

yolov4-obj.cfg

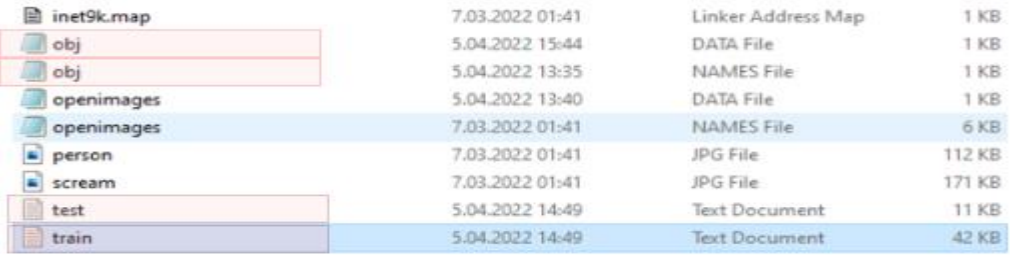
```
[net]
# Testing
#batch=1
#subdivisions=1
# Training
batch=64
subdivisions=64
width=416
height=416
channels=3
momentum=0.949
decay=0.0005
angle=0
saturation = 1.5
exposure = 1.5
hue=.1

learning_rate=0.001
burn_in=1000
max_batches = 4000
policy=steps
steps=3200,3600
scales=.1,.1
```

Şekil 3.16. Yolov4-obj.cfg dosyasındaki ayarlanacak parametreler

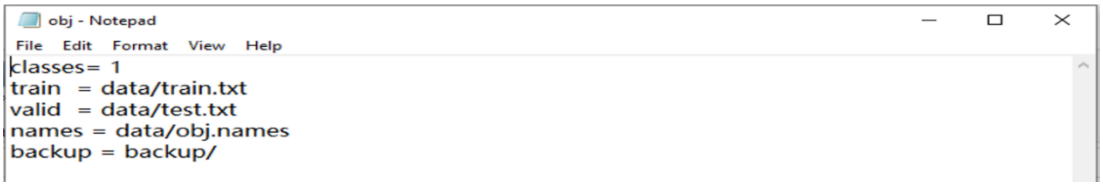
```
[convolutional]
size=1
stride=1
pad=1
filters=18
activation=linear


[yolo]
mask = 0,1,2
anchors = 12, 16, 19, 36, 40, 28, 36, 75, 76, 55, 72, 146, 142, 110, 192, 243, 459, 401
classes=1
num=9
jitter=.3
ignore_thresh = .7
truth_thresh = 1
```

Şekil 3.17. Yolov4-obj.cfg dosyasındaki ayarlanacak parametrelerin devamı

Şekil 3.18.'de gösterildiği gibi Darknet'in "data" dosyasında "obj" adında names ve data uzantılı iki dosya oluşturarak bu dosyalara modelimizle ilgi sınıf adı, sınıf sayısı, eğitim, doğrulama seti ve modelin eğitim sırasında oluşturduğu ağırlıkların yedekleme adreslerini kaydediyoruz. Şekil 3.19. ve Şekil 3.20.'de oluşturulacak dosyanın içerikleri gösterilmektedir.



Şekil 3.18. Darknet'te oluşturulacak obj, train ve test dosyaları



Şekil 3.19. obj.data dosyası

Şekil 3.20. obj.names dosyası

Şekil 3.18.'de gösterildiği gibi modelimizin eğitim ve doğrulama veri setindeki görüntülerin adreslerinin bulunduğu train ve test dosyamızı oluşturulmaktadır. Şekil 3.22. (a) 'da train.txt dosyasındaki görüntü adresleri ve Şekil 3.22. (b) 'de test.txt dosyasındaki görüntü içerikleri gösterilmektedir.

```
with open("train.txt",'r+') as files:
    for i in range(1,809):
        files.writelines(f'data/obj/{i}.jpg\n')
    for i in range(1010,2610):
        files.writelines(f'data/obj/{i}.jpg\n')
```

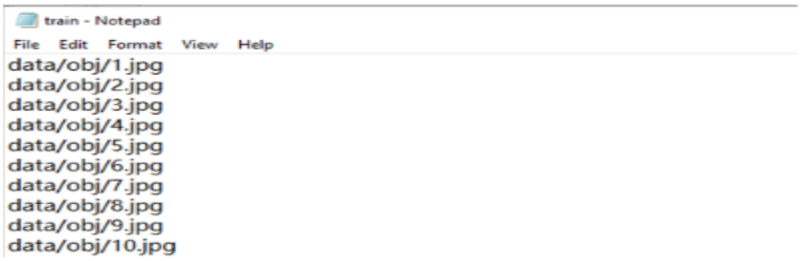
(a)

```
with open("test.txt",'r+') as files:
    for i in range(809,1010):
        files.writelines(f'data/obj/{i}.jpg\n')
    for i in range(2610,3010):
        files.writelines(f'data/obj/{i}.jpg\n')
```

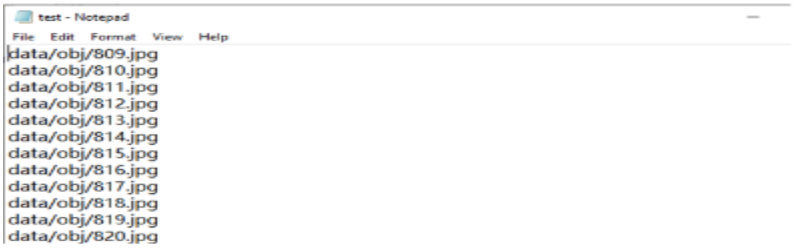
(b)

Şekil 3.21. Train (a) ve Test (b) setlerinin adreslerinin oluşturulması

Şekil 3.21. (a) 'da bu dosyadaki içerikleri oluşturmak için Şekil 3.21. (a) 'da gösterilen Python ile yazılmış kodu kullanarak eğitim setindeki görüntülerin adreslerini ve Şekil 3.21. (b) 'de gösterilen kod ile doğrulama setindeki görüntülerin adreslerini oluşturmaktayız.



(a)



(b)

Şekil 3.22. Train (a) ve Test (b) dosyalarını içerikleri

Son olarak Darknet'in Github adresinden evrişimsel katmanları için önceden eğitilmiş YOLOv4 ağırlığı indirmemiz gerekmektedir. Dosyanın adı ise yolov4.conv.137 [73].

3.7. Modelin Eğitimi

Konfigürasyon işlemlerini tamamlandıktan sonra model YOLOv4 ağırlığı ile eğitilmiştir. Eğitim işlemi CUDA yazılım konfigürasyonunun yapıldığı laptop üzerinden yapılmış olup, modelin eğitildiği laptoptun donanım bilgileri:

- İşletim Sistemi: Windows 10 Enterprise (Versiyon 21H2)
- CPU: İntel(R) Core(TM) i7-4720HQ CPU@ 2.60GHz 2.60GHz
- RAM: 8.0 GB
- GPU: NVİDİA GeForce GTX 950M, Özel GPU belleği: 4GB, Paylaşılan Bellek: 4GB

Modelin eğitimi Darknet'in "darknet.exe detector" ile başlayan birkaç parametreden oluşan komut satır ile eğitilmektedir. Bu parametreler modeldeki veri setinin bilgileri tutulan dosyayı, önceden eğitilmiş YOLOv4 ağırlıklarından oluşmaktadır. Şekil 3.23.'te modeli eğitmek için kullanılan komut satırı gösterilmektedir.

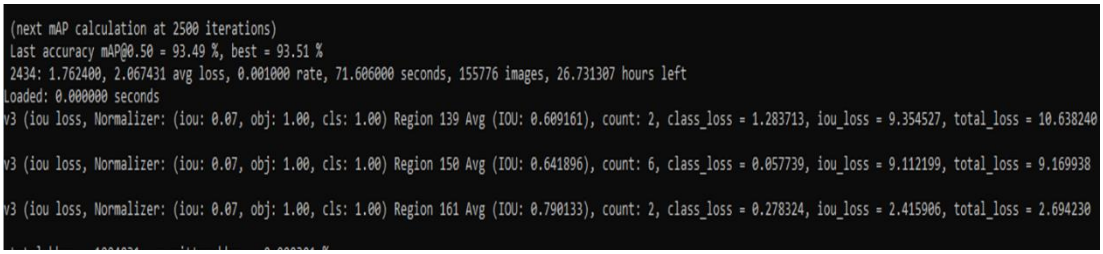
```
darknet.exe detector train data/obj.data cfg/yolov4-obj.cfg yolov4.conv.137 -map
```

1 2 3 4 5

1. Modelin eğitimi için kullanılan değer
2. Veriler ile ilgili bilgilerin tutulduğu dosya
3. Modelin yapısal bilgilerin tutulduğu konfigürasyon dosyası
4. Eğitim için önceden eğitilmiş YOLOv4 ağırlığı
5. Map parametresi Ortalama Hassasiyeti (mAP) verir

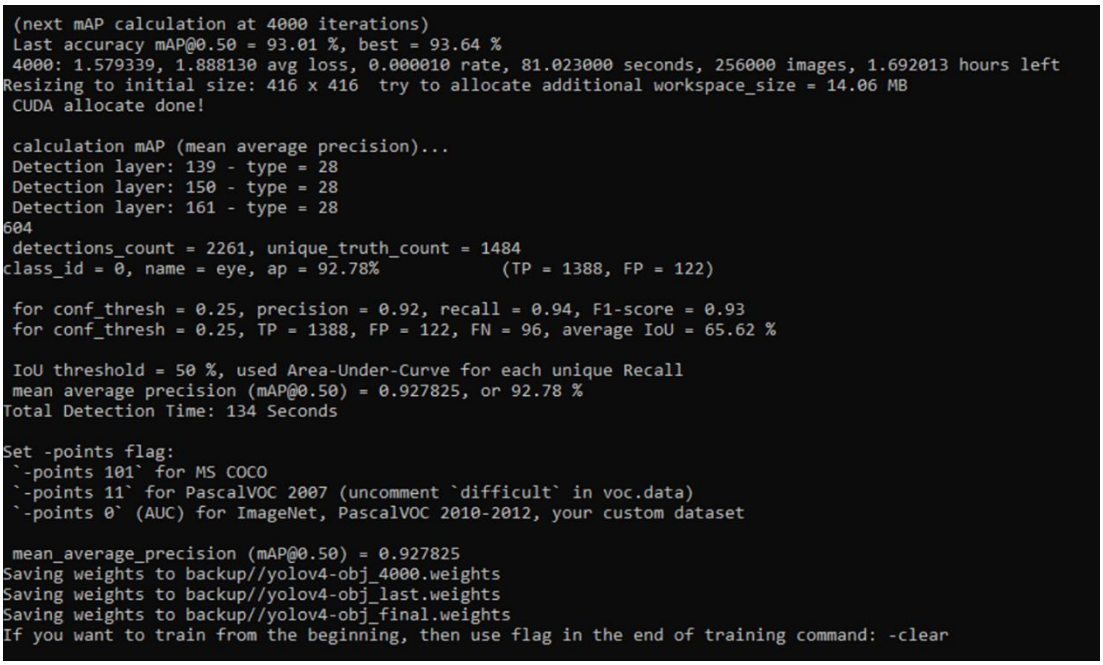
Şekil 3.23. Modelin eğitimi için kullanılan komut satırı

Veri setindeki görüntülerin eğitimi 4000 epoch eğitilmiştir ve modelin eğitimi yaklaşık 54 saat sürmüştür. Şekil 3.24.'te modelin 2434.epoch'takı verilerini göstermektedir. Bu verilerde mAP@0.50 değeri, en iyi Ortalama Kesinlik Ortalaması (mean Average Pricision, mAP), ortalama kayıp değeri (Average Loss), öğrenim onaranı (Learning Rate), eğitim süresi, kalan süre gibi bilgileri göstermektedir.



Şekil 3.24. Modelin 2434.iterasyonu için eğitim bilgileri

Modelin eğitimi tamamlandıktan sonra Şekil 3.25.'te modelin eğitimindeki Ortalama Kesinlik (AP), TP, FP, FN, Kesinlik (Precision), Hassasıyet (Recall), Ortalama Kesinlik Ortalaması (mAP), F1-puanı (F1-score), ortalama IOU gibi performans parametre değerleri gösterilmektedir.



Şekil 3.25. Modelin eğitimi tamamlandıktan sonra ulaşılan sonuçlar

3.8. Modelin Eğitim ve Test Sonuçlarının Değerlendirmesi

Eğitimini tamamladığımız modelin kesinlik, hassasiyet, ortalama kesinlik, ortalama kesinlik ortalaması gibi değerlerini model eğitiminin sonunda elde ettiğimiz veriler ile gösterilmektedir.

3.8.1. Modelin performans değerlendirme metrikleri

Derin öğrene problemlerinde modelimizin başarısını ifade edebilmek için çeşitli ölçütler mevcuttur. Keskinlik, hassasiyet ve ortalama kesinlik gibi ölçütler modelin başarımını ölçmek için kullanılmaktadır. Modelin performans değerlendirmesinde kullanılanlar;

- Hassasiyet (Recall): görüntüdeki kaç tane pozitif örneğin doğru tahmin edildiğini gösterir. Denklem 3.5'te gösterildiği gibi hesaplanmaktadır

$$Hassasiyet = \frac{TP}{TP+FN} \tag{3.5}$$

- Kesinlik (Precision): Kesinlik oranı tahmin sonucu içindir, tahmin edilen pozitif numunelerin kaçının gerçek pozitif numuneler olduğunu gösterir ve Denklem 3.6'de verilen formülle hesaplanmaktadır.

$$Kesinlik = \frac{TP}{TP+FP} \tag{3.6}$$

- Ortalama Kesinlik (AP): Temel olarak kesinlik-geri çağırma eğrisinin altında kalan alandır. Bu, nesne tanıma problemlerinin doğruluğunu ölçmede popüler bir ölçümdür. Ağırlığın hassasiyetteki artış olduğu her bir eşikteki kesinliklerin ağırlıklı toplamı olarak da ifade edilebilir (Denklem 3.7). Denklemde verilen Rn ve Pn güven eşiğindeki kesinlik ve hassasiyeti ifade etmektedir.

$$AP = \sum_0^n (R_n - R_{n-1}) P_n \tag{3.7}$$

- F1-puanı (F1-Score), harmonik ortalamalarını alarak hatırlama ve kesinliği birleştiren bir ölçümdür (Denklem 3.8). Denklemdeki P ise kesinliği, R ise hassasiyeti ifade etmektedir.

$$F1 = 2\left(\frac{P \cdot R}{P+R}\right) \tag{3.8}$$

– Ortalama Kesinlik Ortalaması (mAP) AP'nin ortalamasıdır. Bazı bağlamlarda, AP her sınıf için hesaplanır ve mAP'yi elde etmek için ortalaması alınır. Ortalama Ortalama Hassasiyet veya mAP puanı, mevcut farklı algılama zorluklarına bağlı olarak tüm sınıflar ve/veya genel IoU eşikleri üzerinden ortalama AP alınarak hesaplanır (Denklem 3.9).

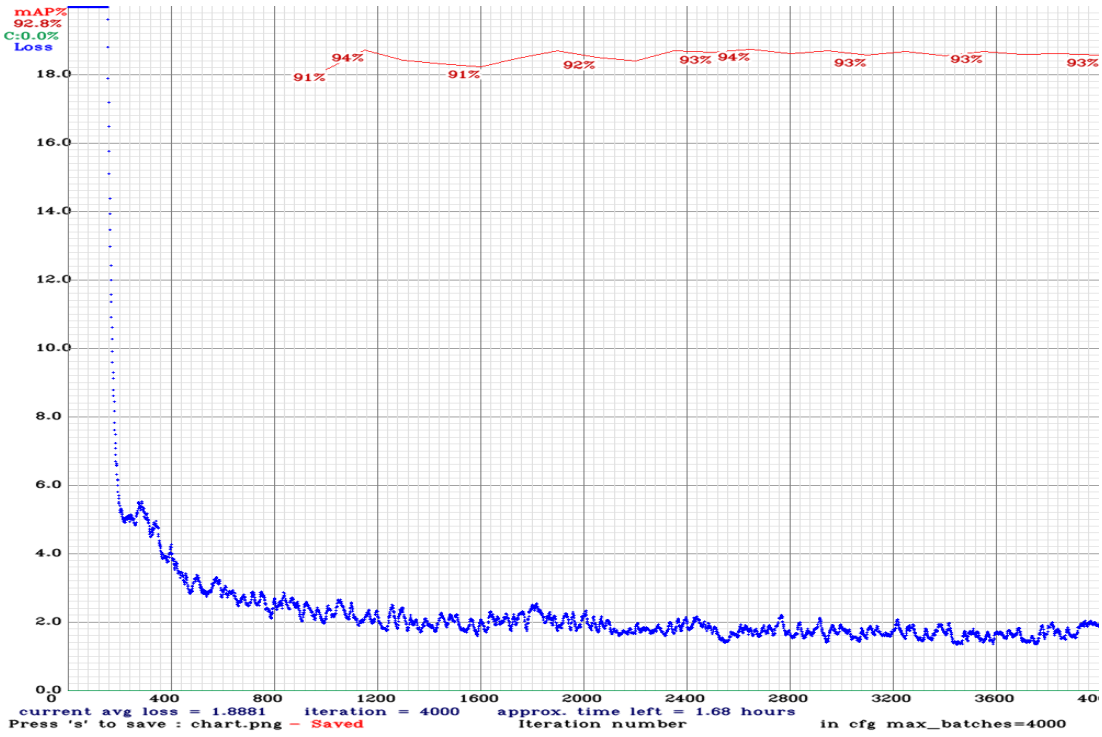
$$mAP = \frac{1}{n} \sum_{i=0}^{n} AP_i \tag{3.9}$$

– Kesiştirilmiş Bölgeler (IoU): İoU, öngörülen sınırlayıcı kutu ile gerçek durum sınırlayıcı kutunun kesişme alanı ve birleşim alanının oranıyla verilir (Denklem 3.10).

$$\text{İ}oU = \frac{\text{Örtüşmüş } Alan}{Birleştirilmiş\ Alan} = \frac{|A \cap B|}{|A \cup B|} \tag{3.10}$$

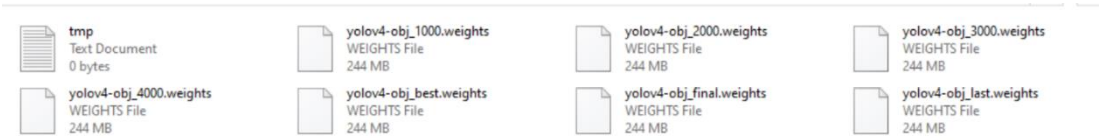
3.8.2. Modelin performans analizi

Modelin eğitimi 4000 epoch için yaklaşık 54 saat sürdürülerek modelin performansını ölçen ölçekler hesaplanmıştır. Eğitim sonunda elde edilen parametrelere göre modelin ortalama kaybı, hataların zamanla azalması ve buna bağlı olarak ortalama kesinlik ortalaması ve ortalama kesinliğin artması modelin başarısı için önem taşımaktadır. Şekil 3.26.'dan görüleceği gibi ortalama kayıplar model eğitimin ilk epoch'larında dörttün üzerinde seyrederken, eğitimin sonlarına doğru ortalama 1.8'e düşürülmüştür. Ortlama kayıp ve hata miktarlarının epoch sayısını artmasıyla düştüğü gözlemlenmektedir ve bu düşüşlerin modelin testinde iyi performans göstermesini sağlamaktadır.



Şekil 3.26. Modelin eğitim aşaması performans grafiği

Model eğitimi sırasında, her 1000 epoch'ta bir kez ağın ağırlığını kaydedilmektedir ve eğitimin sonuna doğru modelin 1000'lık, 2000'lık, 3000'lık, 4000'lık, modelin en iyi performans sergileyen ağırlığı, final ağırlık ve eğitimin sonundaki ağırlık olmak üzere farklı ağırlıklar modelin yedekleme dosyasında oluşmaktadır (Şekil 3.27.).

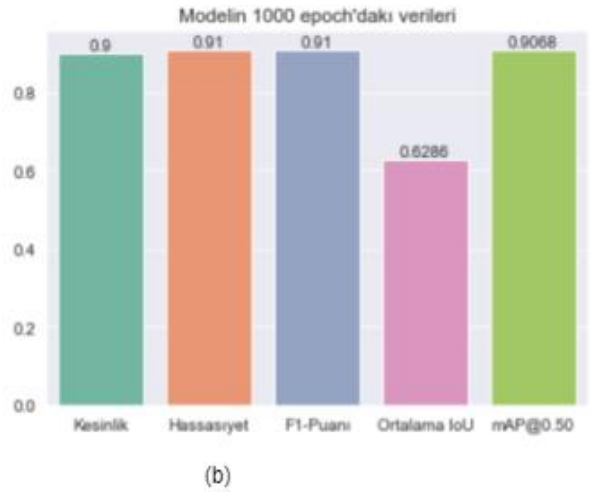


Şekil 3.27. Modelin eğitimi sırasında oluşan ağırlıklar

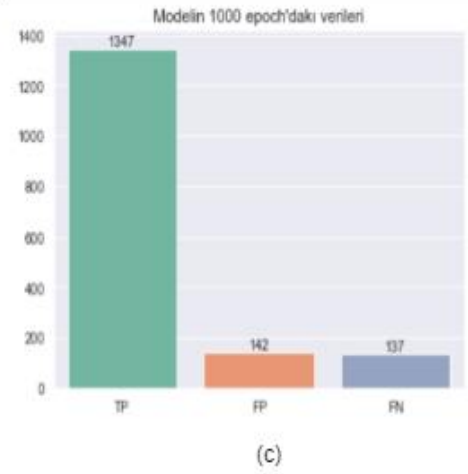
Modelin ilk 1000 epoch kaydettiği verilere göre ağın kesinliği 0.90, hassasiyeti 0.91, F1-puanı 0.91 olmuştur (Şekil 3.28. (a) ve (b)). Modelin TP değeri 1347, FP değeri 142, FN değeri 137 (Şekil 3.28. (a) ve (c)) ve ortalama IoU değeri ise %62,86 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %90,68 olarak hesaplanmaktadır (Şekil 3.28. (a) ve (b)).

(a)

```
 detections_count = 5170, unique_truth_count = 1484
class_id = 0, name = eye, ap = 90.68%           (TP = 1347, FP = 142)

 for conf_thresh = 0.25, precision = 0.90, recall = 0.91, F1-score = 0.91
 for conf_thresh = 0.25, TP = 1347, FP = 142, FN = 137, average IoU = 62.86 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.906830, or 90.68 %
```



(b)



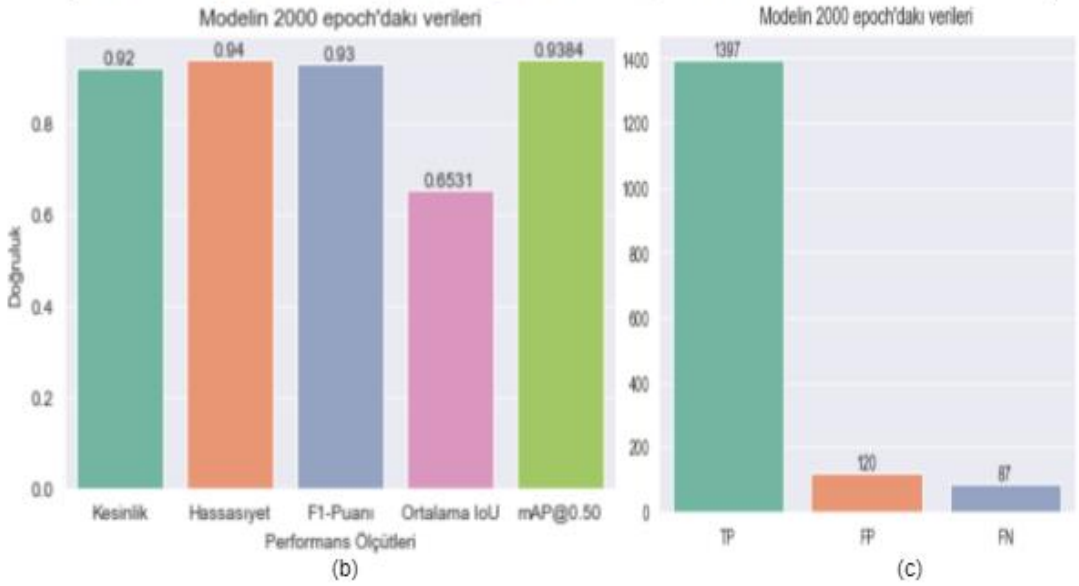
(c)

Şekil 3.28. Modelin ilk 1000 epoch'dakı performans değerleri

Modelin 2000 epoch'ta kaydettiği verilere göre ağın ortalama kesinliği %93,84, kesinlik 0.92, hassasiyeti 0.94, F1-puanı 0.93 olmuştur (Şekil 3.29. (a) ve (b)). Modelin TP değeri 1397, FP değeri 120, FN değeri 87 (Şekil 3.29. (a) ve (c)) ve ortalama IoU değeri ise %65,31 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %93,84 olarak hesaplanmaktadır (Şekil 3.29. (a) ve (b)).

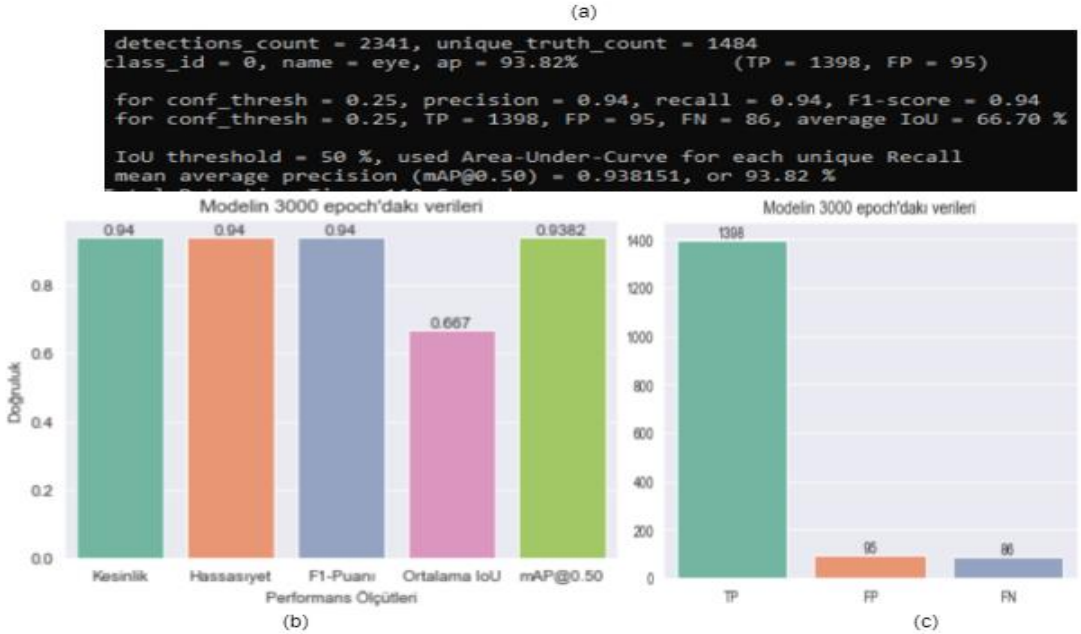
(a)

```
 detections_count = 2897, unique_truth_count = 1484
class_id = 0, name = eye, ap = 93.84%           (TP = 1397, FP = 120)

 for conf_thresh = 0.25, precision = 0.92, recall = 0.94, F1-score = 0.93
 for conf_thresh = 0.25, TP = 1397, FP = 120, FN = 87, average IoU = 65.31 %

 IoU threshold = 50 %, used Area-Under-Curve for each unique Recall
 mean average precision (mAP@0.50) = 0.938435, or 93.84 %
```



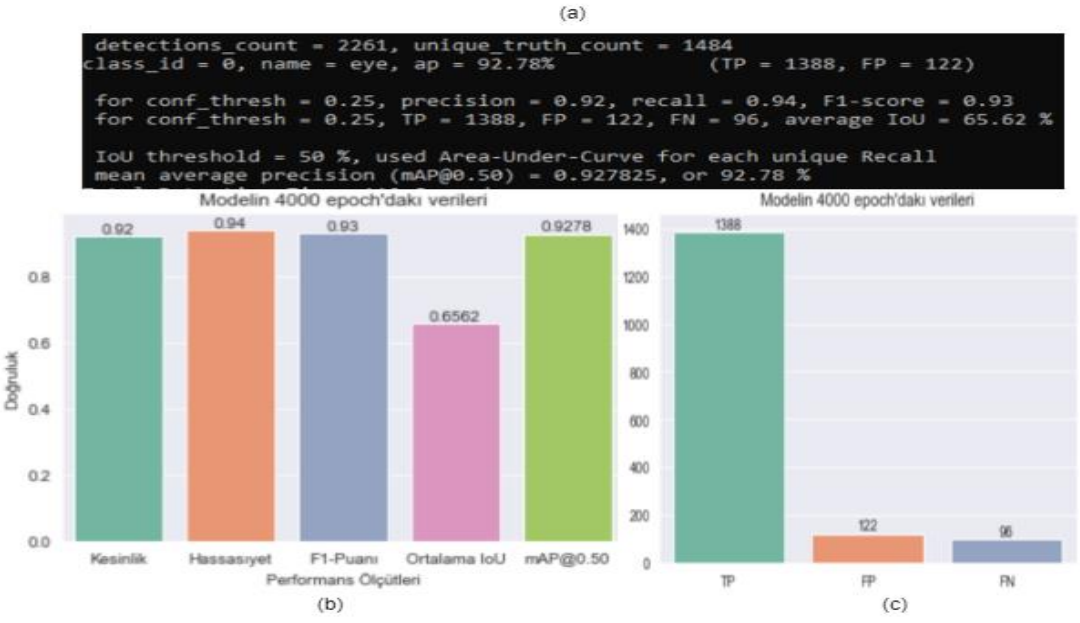
Şekil 3.29. Modelin 2000 epoch'ta kaydettiği performans değerleri

Modelin 3000 epoch'ta kaydettiği verilere göre ağın ortalama kesinliği %93,82, kesinlik 0.94, hassasiyeti 0.94, F1-puanı 0.94 olmuştur (Şekil 3.30. (a) ve (b)). Modelin TP değeri 1398, FP değeri 95, FN değeri 86 (Şekil 3.30. (a) ve (c)) ve ortalama IoU değeri ise %66,70 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %93,82 olarak hesaplanmaktadır (Şekil 3.30. (a) ve (b)).



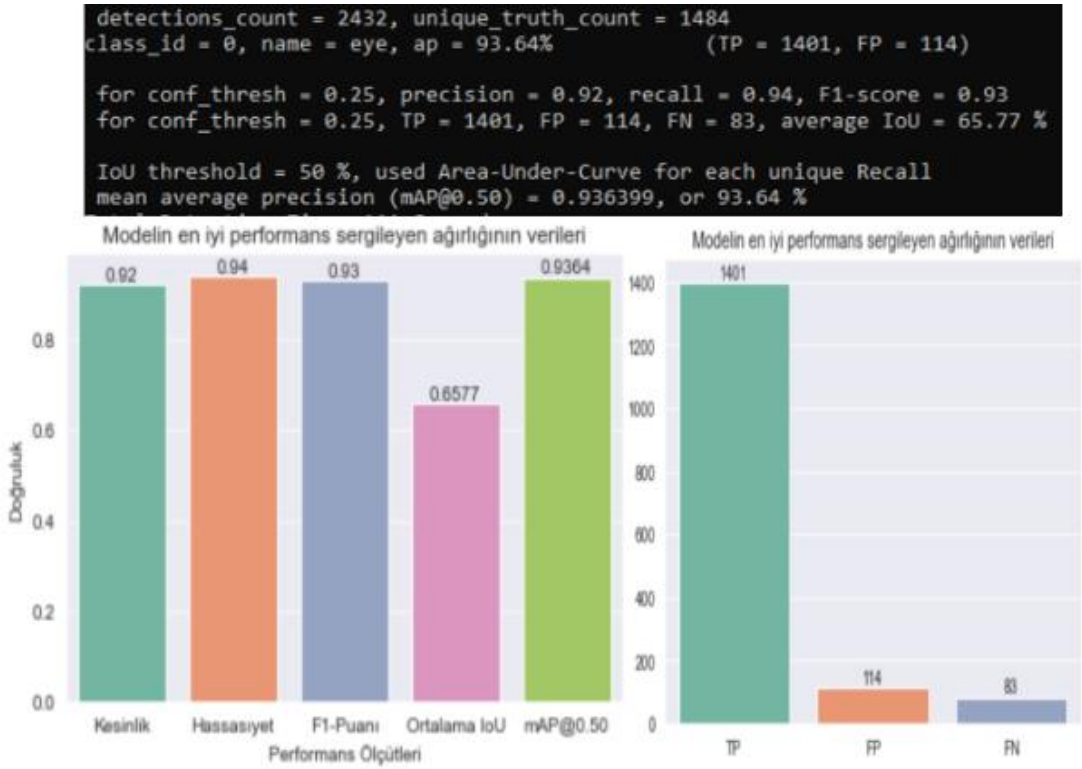
Şekil 3.30. Modelin 3000 epoch'ta kaydettiği performans değerleri

Modelin 4000 epoch'ta kaydettiği verilere göre ağın ortalama kesinliği %92,78, kesinlik 0.92, hassasiyeti 0.94, F1-puanı 0.93 olmuştur (Şekil 3.31. (a) ve (b)). Modelin TP değeri 1388, FP değeri 122, FN değeri 96 (Şekil 3.31. (a) ve (c)) ve ortalama IoU değeri ise %65,62 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %92,78 olarak hesaplanmaktadır (Şekil 3.31. (a) ve (b)).



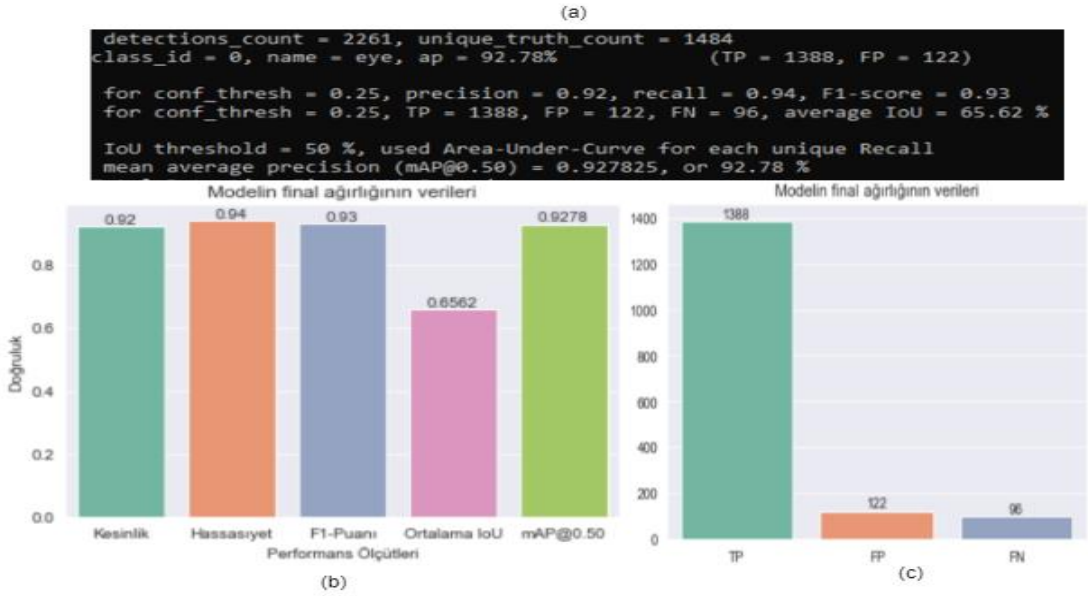
Şekil 3.31. Modelin 4000 epoch'ta kaydettiği performans değerleri

Modelin en iyi ağırlık verilerine göre ağın ortalama kesinliği %93,64, kesinlik 0.92, hassasiyeti 0.94, F1-puanı 0.93 olmuştur (Şekil 3.32.). Modelin TP değeri 1401, FP değeri 114, FN değeri 83 (Şekil 3.32.) ve ortalama IoU değeri ise %65,77 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %93,64 olarak hesaplanmaktadır (Şekil 3.32.).



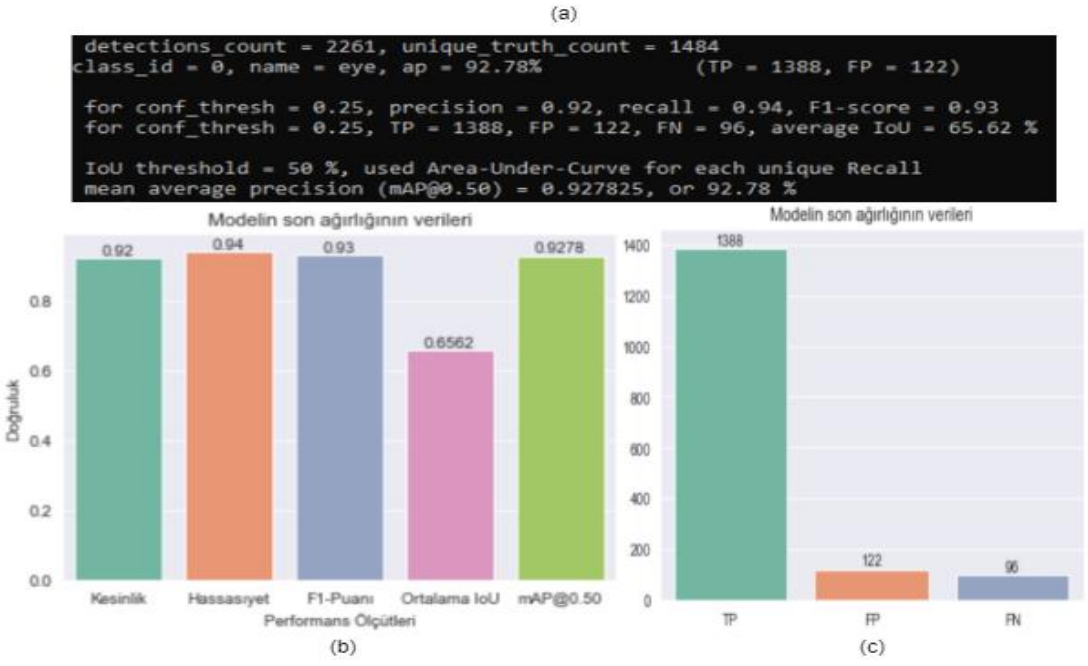
Şekil 3.32. Modelin en iyi performans sergileyen ağırlığının verileri

Modelin final ağırlık verilerine göre ağın ortalama kesinliği %92,78, kesinlik 0.92, hassasiyeti 0.94, F1-puanı 0.93 olmuştur (Şekil 3.33. (a) ve (b)). Modelin TP değeri 1388, FP değeri 122, FN değeri 96 (Şekil 3.33. (a) ve (c)) ve ortalama IoU değeri ise %65,62 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %92,78 olarak hesaplanmaktadır (Şekil 3.33. (a) ve (b)).



Şekil 3.33. Modelin final ağırlığının verileri

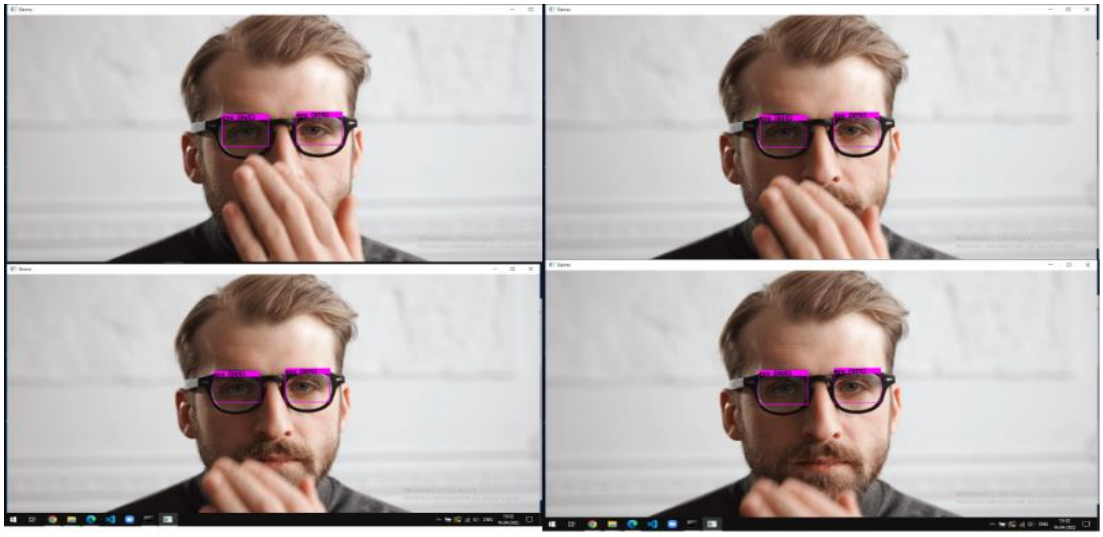
Modelin son ağırlık verilerine göre ağın ortalama kesinliği %92,78, kesinlik 0.92, hassasiyeti 0.94, F1-puanı 0.93 olmuştur (Şekil 3.34. (a) ve (b)). Modelin TP değeri 1388, FP değeri 122, FN değeri 96 (Şekil 3.34. (a) ve (c)) ve ortalama IoU değeri ise %65,62 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 ise %92,78 olarak hesaplanmaktadır (Şekil 3.34. (a) ve (b)).



Şekil 3.34. Modelin son ağırlığının performans verileri

Model, eğitim sırasında kullanılmamış farklı görüntü ve videolar ile test edilmiştir. Model testinde kullanılan görüntüler açık erişimli web sitesi olan Pexels'den [74] indirilmiştir. Şekil 3.35.'te tahmin edilen görüntülerin konumları, sınıfı ve olasılıkları gösterilmektedir. Şekil 3.36.'da test edilen video görüntülerin ekran görüntü verilmektedir ve Şekilde görüntünün konumları, sınıfı ve olasılıkların tahminlerini göstermektedir.

Şekil 3.35. Modelin test sonuçlari



Şekil 3.36. Modelin video üzerindeki gerçek zamanlı test sonuçları

3.8.3. Göz Koruma kontrolü modelinin performans analizi

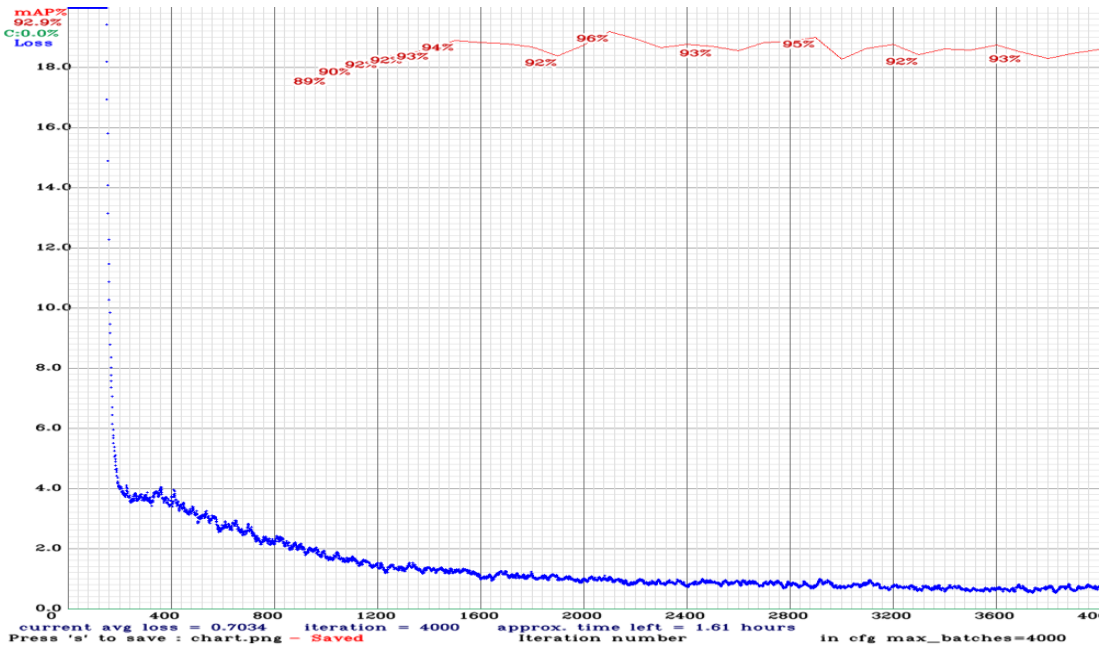
İş Güvenliği açısından göz koruma kontrolü yapan model için hazırlanmış verilerimiz işlendikten sonra (Şekil 3.37.), modelimizi 4000 iterasyon çalıştırılmıştır. Modelin konfigürasyon işlemleri daha önceki bölümlerde yaptiğimiz konfigürasyon işlemler ile aynıdır. Sadece modelin sınıf ve fıltre sayısı daha önceki modele göre faklılık göstermektedir. Bu modelin sınıf sayısı korumalı ve korumasız olarak iki farklı sınıftan oluşmuştur ve filtre sayisi Denklem 3.4'e göre 21'dir.

Şekil 3.37. Göz koruma kontrolü veri setinden örnekler

Modelin en iyi ağırlık verilerine göre ağın kesinliği 0. 93, recall 0.91, F1-puanı 0.92 olmuştur. Modelin TP değeri 239, FP değeri 19, FN değeri 24 ve ortalama IoU değeri ise %70,28 olmuştur. Ağın ortalama kesinlik ortalaması mAP@0.50 %95,88 olarak hesaplanmaktadır (Şekil 3.38.). Sınıf_0'ın (Korumalı Gözlüklü) ortalama kesinliği %93,88, TP değeri 96, FP değeri 10 olmuştur. Sınıf_1'in (Korumalı Gözlüksüz) ortalama kesinliği %97,89, TP değeri 143, FP değeri 9 olmuştur.

Şekil 3.38. Göz Koruma kontrol modelini en iyi performans ağırlığının değerleri

Şekil 3.39.'da modelin eğitimi sırasında kaydedilen genel performans değerleri gösterilmektedir. Tablo 3.1.'de gösterildiği gibi modelin en iyi kesinlik, hassasiyet, F1-puan ve IoU değerleri modelin 4000 iterasyonlu ağırlığında görülmektedir ve modelin en iyi ortalama kesinlik ortalaması modelin en iyi ağırlık veren modelinde görülmektedir.



Şekil 3.39. Koruma kontrolü modelinin performans grafığı

Tablo 3.2. Modellerin performans karşılaştırılması

Model	Kesinlik	Hassasıyet	F1-puan	IoU	mAP@0.5
Yolov4-obj_1000.weights	0.71	0.81	0.75	50.31%	88.77%
Yolov4-obj_2000.weights	0.92	0.90	0.91	69.52%	91.35%
Yolov4-obj_3000.weights	0.92	0.91	0.92	66.70%	93.82%
Yolov4-obj_4000.weights	0.94	0.93	0.94	71.86%	92.88%
Yolov4-obj_best.weights	0.93	0.91	0.92	70.28%	95.88%

Şekil 3.40.'te modelin test sonuçları gösterilmektedir. Test sonuçlarına göre model koruma gözlüğü ile normal gözlüğü ayrıt edebilmektedir.

Şekil 3.40. Göz koruma kontrol modelinin test sonuçları

BÖLÜM 4. SONUÇLAR VE GELECEK ÇALIŞMALAR

Göz algılama, yüz tanıma işlemleri, bakış takibi, bankacılık, adliye ve sağlık gibi birçok alanlarda çok öneme sahip bir alandır. Bu tez çalışmasında yapay zekâ teknikleri kullanarak haraketli görüntülerde yüz görüntülerinden göz tespiti çalışması yapılmıştır. Görüntülerden göz tespiti için derin öğrenme yöntemlerindeki küçük nesnelerin tespitinde doğruluk, algılama ve çıkarım hizi açısından başarımı yüksel olan tek basamaklı nesne algılama modeli olan YOLOv4 algoritması kullanılmıştır. LFW ve FDDB veri setinden alınmış görüntüler OpenCV ile boyutlandırma işlemi yapıldıktan sonra MakeSense ile görüntülerdeki gözlerin sınırlayıcı kutulara alınma işlemi yapılmıştır. Ön işlemeden geçen görüntüler eğitim ve doğrulama veri setlerine ayrılarak modelin eğitimine hazırlanılmıştır. Daha sonra gerekli yazılımların konfigürasyon işlemleri yapıldıktan sonra görüntüler modele girdi olarak verilmiştir ve yaklaşık 54 saatlik eğitim sürecinden sonra göz tespiti için modelimiz eğitilerek tespit işlemlerine hazır hale getirilmiştir.

Göz koruyucu kontrolü çalışmasında, internet üzerinden elde edilmiş farklı iş ortamında çekilmiş koruyucu gözlük kullanımı görüntüleri Derin Öğrenme ile saptanmıştır. Görüntüleri modele uygun biçime getirmek için Python ve OpenCV kütüphanesi kullanarak görüntülerin boyutları ve formatları ayarlanmıştır. Nesne tanıma problemleri için görüntü işleme aracı olan internet tabanlı MakeSence yazılımı ile etiketleme işlemi gerçekleştirilmiştir. MakeSense her fotoğraf için etiket sınıfına ait koordinat bilgilerini oluşturmuştur. Model Darknet Çerçevesi yardımıyla YOLOv4 Algoritması ile eğitilmiştir. Sonuçlardan modelin doğruluğu görüntüler ile test edilmiştir. YOLOv4 algoritması kullanılarak uygun sonuçlara ulaşılan bu çalışma, iş yerlerinde iş güvenliği ve çalışanların sağlığı açısından gerçek zamanlı ve kolay denetleme aracı geliştirmiştir.

Gelecek çalışmalarda geliştirilen modelin doğruluğunu artırabilmek için farklı aydınlatma, poz gibi özelliğe sahip iyi çözünürlükteki daha fazla görüntüler ile model eğitilebilir. Güçlü hesaplama gücüne sahip GPU ile model eğitilirse modelin gerçek zamanlı nesne tespit hızlıda ona bağlı olarak artar. Bu nedenle modelin daha iyi çıkarım hızını ve daha fazla FPS'i elde etmesi için güçlü GPU ile eğitilebilir. Modelin bilgisayarlarda eğitilmesi için çok iyi donanım gücüne gerek duyduğundan ve yazılım ortamının konfigürasyonu çok karışık olduğundan dolayı ücretli Google Colab [75] (Ücretsiz Google Colab ile günlük eğitilebilecek süre 6 saati geçmemektedir.) kullanarak eğitilmesi sağlanabilir. Koruyucu gözlük kontrol modelinin koruyucu gözlük kullanımın tespiti yüksek bir tespit performansı sergilemektedir ancak çok nadir olarak yüz rengi çok siyah olan koruyucu gözlük kullananları gözlüksüz olarak tespit etmektedir. Bu yanlışı önlemek için modelin eğitimi sırasında kullanılan görüntülerde siyah renkli yüze sahip koruyucu gözlük takan çalışanların fazla olmasına özen gösterilmesi modelin daha sağlıklı çalışması açısından önemli olacaktır.

KAYNAKLAR

[1] S. Karahan and Y. S. Akgul, “Eye detection by using deep learning,” in 2016 24th Signal Processing and Communication Application Conference (SIU), May 2016, pp. 2145–2148. doi: 10.1109/SIU.2016.7496197.

[2] K. DONUK, A. ARI, and D. HANBAY, “Yüz İmgelerinden Göz Bölgelerinin Tespitinde ESA Tabanlı Alternatif Bir Yaklaşım,” Fırat Üniversitesi Mühendislik Bilimleri Dergisi, Aug. 2021, doi: 10.35234/fumbd.956120.

[3] M. Yu, Y. Lin, and X. Wang, “An efficient hybrid eye detection method,” Turkish Journal of Electrical Engineering and Computer Sciences, vol. 24, no. 3, pp. 1586–1603, 2016, doi: 10.3906/elk-1312-150.

[4] Y. Jia et al., “Caffe: Convolutional Architecture for Fast Feature Embedding,” in Proceedings of the 22nd ACM international conference on Multimedia, Nov. 2014, pp. 675–678. doi: 10.1145/2647868.2654889.

[5] “NVIDIA DIGITS | NVIDIA Developer.” https://developer.nvidia.com/digits., Erişim Tarihi: 16.04.2022.

[6] V. Jain and E. Learned-Miller, “FDDB: A Benchmark for Face Detection in Unconstrained Settings.” [Online]. Available: http://news.yahoo.com

[7] B.-C. Chen, C.-S. Chen, and W. H. Hsu, “Cross-Age Reference Coding for Age-Invariant Face Recognition and Retrieval,” 2014, pp. 768–783. doi: 10.1007/978-3-319-10599-4_49.

[8] S. Karahan and Y. S. Akgul, “Eye detection by using deep learning,” in 2016 24th Signal Processing and Communication Application Conference (SIU), May 2016, pp. 2145–2148. doi: 10.1109/SIU.2016.7496197.

[9] E. Civik and U. Yuzgec, “Deep Learning Based Continuous Real-Time Driver Fatigue Detection for Embedded System,” Oct. 2020. doi: 10.1109/SIU49456.2020.9302035.

[10] “Karayolu Trafik Kaza İstatistikleri, 2020,” 2021. [Online]. Available: https://data.tuik.gov.tr/Bulten/Index?p=Road-Traffic-Accident-Statistics-2020-37436., Erişim Tarihi: 10.12.2021.

[11] H. Nguyen Quoc and V. Truong Hoang, “Real-Time Human Ear Detection Based on the Joint of Yolo and RetinaFace,” Complexity, vol. 2021, 2021, doi: 10.1155/2021/7918165.

[12] H. Aung, A. v. Bobkov, and N. L. Tun, “Face detection in real time live video using yolo algorithm based on VGG16 convolutional neural network,” in Proceedings - 2021 International Conference on Industrial Engineering, Applications and Manufacturing, ICIEAM 2021, 2021, pp. 697–702. doi: 10.1109/ICIEAM51226.2021.9446291.

[13] D. Garg, P. Goel, S. Pandya, A. Ganatra, and K. Kotecha, “A Deep Learning Approach for Face Detection using YOLO,” in 2018 IEEE Punecon, Nov. 2018, pp. 1–4. doi: 10.1109/PUNECON.2018.8745376.

[14] R. Madura Meenakshi, N. Padmapriya, N. Venkateswaran, R. Ravikumar, and R. Chelliah, “Localization of Eye Region in Infrared Thermal Images using Deep Neural Network,” in 2021 International Conference on Wireless Communications, Signal Processing and Networking, WiSPNET 2021, Mar. 2021, pp. 446–450. doi: 10.1109/WiSPNET51692.2021.9419446.

[15] M. Yu, Y. Lin, and X. Wang, “An efficient hybrid eye detection method,” Turkish Journal of Electrical Engineering and Computer Sciences, vol. 24, no. 3, pp. 1586–1603, 2016, doi: 10.3906/elk-1312-150.

[16] J. Patterson and A. Gibson, “Deep Learning,” Sebastopol, 2017.

[17] N. Buduma, “Fundamentals of Deep Learning,” 2017.

[18] N. Ketkar, Deep Learning with Python. Apress, 2017. doi: 10.1007/978-1-4842-2766-4.

[19] N. K. Manaswi, Deep Learning with Applications Using Python. Apress, 2018. doi: 10.1007/978-1-4842-3516-4.

[20] F. Chollet, “Deep Learning with Python,” 2018.

[21] T. Hope, Y. S. Resheff, and I. Lieder, “Learning TensorFlow,” 2017.

[22] S. Pattanayak, Pro Deep Learning with TensorFlow. Apress, 2017. doi: 10.1007/978-1-4842-3096-1.

[23] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, “Gradient-based learning applied to document recognition,” Proceedings of the IEEE, vol. 86, no. 11, pp. 2278–2323, 1998, doi: 10.1109/5.726791.

[24] A. Krizhevsky, I. Sutskever, and G. E. Hinton, “ImageNet classification with deep convolutional neural networks,” Commun ACM, vol. 60, no. 6, pp. 84–90, Jun. 2017, doi: 10.1145/3065386.

[25] "7.1. Derin Evrişimli Sinir Ağları (AlexNet) — Derin Öğrenmeye Dalış 0.17.2 documentation." http://preview.d2l.ai/d2l-tr/master/chapter_convolutional-modern/alexnet.html., Erişim Tarihi: 23.03.2022.

[26] K. Simonyan and A. Zisserman, "Very Deep Convolutional Networks for Large-Scale Image Recognition," Sep. 2014, [Online]. Available: http://arxiv.org/abs/1409.1556

[27] T. Bezdan and N. Bačanin Džakula, "Convolutional Neural Network Layers and Architectures," pp. 445–451, May 2019, doi: 10.15308/SINTEZA-2019-445-451.

[28] C. Szegedy et al., "Going deeper with convolutions," Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, vol. 07-12-June-2015, pp. 1–9, Oct. 2015, doi: 10.1109/CVPR.2015.7298594.

[29] Z. Zou, Z. Shi, Y. Guo, and J. Ye, "Object Detection in 20 Years: A Survey," May 2019.

[30] J. U. Kim and Y. Man Ro, "Attentive Layer Separation for Object Classification and Object Localization in Object Detection," in 2019 IEEE International Conference on Image Processing (ICIP), Sep. 2019, pp. 3995–3999. doi: 10.1109/ICIP.2019.8803439.

[31] K. E. A. van de Sande, J. R. R. Uijlings, T. Gevers, and A. W. M. Smeulders, "Segmentation as selective search for object recognition," Proceedings of the IEEE International Conference on Computer Vision, pp. 1879–1886, 2011, doi: 10.1109/ICCV.2011.6126456.

[32] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich Feature Hierarchies for Accurate Object Detection and Semantic Segmentation," in 2014 IEEE Conference on Computer Vision and Pattern Recognition, Jun. 2014, pp. 580–587. doi: 10.1109/CVPR.2014.81.

[33] R. Girshick, "Fast R-CNN," in 2015 IEEE International Conference on Computer Vision (ICCV), Dec. 2015, pp. 1440–1448. doi: 10.1109/ICCV.2015.169.

[34] K. He, X. Zhang, S. Ren, and J. Sun, "Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition," IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 37, no. 9, pp. 1904–1916, Sep. 2015, doi: 10.1109/TPAMI.2015.2389824.

[35] "Nesne Tanıma Algoritmaları: R-CNN, Fast R-CNN ve Faster R-CNN Nedir? - Teknoloji.org." https://teknoloji.org/nesne-tanima-algoritmalari-r-cnn-fast-r-cnn-ve-faster-r-cnn-nedir/?msclkid=969848b6ad0a11ec94365c1a3a293983., Erişim Tarihi: 026.03.2022.

[36] “One-stage object detection.” https://machinethink.net/blog/object-detection/., Erişim Tarihi: 27.03.2022.

[37] W. Liu et al., Computer Vision – ECCV 2016, vol. 9905. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-46448-0.

[38] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You Only Look Once: Unified, Real-Time Object Detection,” in 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jun. 2016, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[39] Y. LIU and W. JIANG, “Detection of wearing safety helmet for workers based on YOLOv4,” in 2021 International Conference on Computer Engineering and Artificial Intelligence (ICCEAI), Aug. 2021, pp. 83–87. doi: 10.1109/ICCEAI52939.2021.00016.

[40] H. Wang and H. Yu, “Traffic Sign Detection Algorithm based on improved YOLOv4,” in 2020 IEEE 9th Joint International Information Technology and Artificial Intelligence Conference (ITAIC), Dec. 2020, pp. 1946–1950. doi: 10.1109/ITAIC49862.2020.9339181.

[41] “CS 230 - Evrişimli Sinir Ağları El Kitabı.” https://stanford.edu/~shervine/l/tr/teaching/cs-230/cheatsheet-convolutional-neural-networks#object-detection., Erişim Tarihi: 30.03.2022.

[42] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, “You only look once: Unified, real-time object detection,” in Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition, Dec. 2016, vol. 2016-December, pp. 779–788. doi: 10.1109/CVPR.2016.91.

[43] A. Bochkovskiy, C.-Y. Wang, and H.-Y. M. Liao, “YOLOv4: Optimal Speed and Accuracy of Object Detection,” Apr. 2020, [Online]. Available: http://arxiv.org/abs/2004.10934

[44] A. M. Roy, R. Bose, and J. Bhaduri, “A fast accurate fine-grain object detection model based on YOLOv4 deep neural network,” Oct. 2021, [Online]. Available: http://arxiv.org/abs/2111.00298

[45] F. Iandola, M. Moskewicz, S. Karayev, R. Girshick, T. Darrell, and K. Keutzer, “DenseNet: Implementing Efficient ConvNet Descriptor Pyramids,” Apr. 2014, [Online]. Available: http://arxiv.org/abs/1404.1869

[46] C.-Y. Wang, H.-Y. M. Liao, I.-H. Yeh, Y.-H. Wu, P.-Y. Chen, and J.-W. Hsieh, “CSPNet: A New Backbone that can Enhance Learning Capability of CNN,” Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.11929

[47] P. Ramachandran, B. Zoph, and Q. v. Le, “Searching for Activation Functions,” Oct. 2017, [Online]. Available: http://arxiv.org/abs/1710.05941

[48] Z. Zheng, P. Wang, W. Liu, J. Li, R. Ye, and D. Ren, “Distance-IoU Loss: Faster and Better Learning for Bounding Box Regression,” Nov. 2019, [Online]. Available: http://arxiv.org/abs/1911.08287

[49] G. Ghiasi, T.-Y. Lin, and Q. v. Le, “DropBlock: A regularization method for convolutional networks,” Oct. 2018, [Online]. Available: http://arxiv.org/abs/1810.12890

[50] Z. Yao, Y. Cao, S. Zheng, G. Huang, and S. Lin, “Cross-Iteration Batch Normalization,” Feb. 2020, [Online]. Available: http://arxiv.org/abs/2002.05712

[51] “Introduction to YOLOv4 Object Detection for Beginners | by Rokas Balsys | Python in Plain English.” https://python.plainenglish.io/introduction-to-yolov4-object-detection-ce272a76fedd., Erişim Tarihi: 02.04.2022.

[52] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for Instance Segmentation,” Mar. 2018, [Online]. Available: http://arxiv.org/abs/1803.01534

[53] T.-Y. Lin, P. Dollar, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature Pyramid Networks for Object Detection,” in 2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Jul. 2017, vol. 2017-January, pp. 936–944. doi: 10.1109/CVPR.2017.106.

[54] “YOLOv4. While object detection matures in the… | by Jonathan Hui | Medium.” https://jonathan-hui.medium.com/yolov4-c9901eaa8e61., Erişim Tarihi: 01.04.2022.

[55] J. Redmon and A. Farhadi, “YOLOv3: An Incremental Improvement,” Apr. 2018, [Online]. Available: http://arxiv.org/abs/1804.02767

[56] “Make Sense.” https://www.makesense.ai/., Erişim Tarihi: 03.04.2022.

[57] Skalski Piotr, “GitHub - SkalskiP/make-sense: Free to use online tool for labelling photos. https://makesense.ai.” https://github.com/SkalskiP/make-sense., Erişim Tarihi: 03.04.2022.

[58] “Home - OpenCV.” https://opencv.org/., Erişim Tarihi: 03.04.2022.

[59] “Artificial Intelligence Computing Leadership from NVIDIA.” https://www.nvidia.com/en-us/?msclkid=6645e6bfb68411ec82e2a1ab0f359393., Erişim Tarihi: 07.04.2022.

[60] A. Kaehler and G. Bradski, “OpenCV 3 COMPUTER VISION IN C++ WITH THE OPENCV LIBRARY,” 2016.

[61] B. W. Chung, Pro Processing for Images and Computer Vision with OpenCV. Berkeley, CA: Apress, 2017. doi: 10.1007/978-1-4842-2775-6.

[62] G. B. Huang, M. Ramesh, T. Berg, and E. Learned-Miller, "Labeled Faces in the Wild: A Database for Studying Face Recognition in Unconstrained Environments", doi: 10.1.1.122.8268.

[63] "Download Visual Studio Tools - Install Free for Windows, Mac, Linux." https://visualstudio.microsoft.com/downloads/., Erişim Tarihi: 07.04.2022.

[64] "CUDA Toolkit 11.6 Update 2 Downloads | NVIDIA Developer." https://developer.nvidia.com/cuda-downloads?target_os=Windows&target_arch=x86_64&target_version=10&target_type=exe_local., Erişim Tarihi: 07.04.2022.

[65] "NVIDIA Developer Program Membership Required | NVIDIA Developer." https://developer.nvidia.com/rdp/cudnn-download., Erişim Tarihi: 07.04.2022.

[66] "Welcome to Python.org." https://www.python.org/?msclkid=df23f14eb68311ecaff67c2db5ea9b3f., Erişim Tarihi: 07.04.2022.

[67] "NumPy." https://numpy.org/?msclkid=ee08b997b68311eca3b09e26b20c7905., Erişim Tarihi: 07.04.2022.

[68] "Download | CMake." https://cmake.org/download/., Erişim Tarihi: 07.04.2022.

[69] "Darknet: Open Source Neural Networks in C." https://pjreddie.com/darknet/., Erişim Tarihi: 08.04.2022.

[70] "CUDA - Wikipedia." https://en.wikipedia.org/wiki/CUDA., Erişim Tarihi: 07.04.2022.

[71] "Overview | CMake." https://cmake.org/overview/., Erişim Tarihi: 08.04.2022.

[72] "GitHub - AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)." https://github.com/AlexeyAB/darknet., Erişim Tarihi: 07.04.2022.

[73] "Train a custom YOLOv4 object detector on Windows | by Techzizou | Geek Culture | Medium." https://medium.com/geekculture/train-a-custom-yolov4-object-detector-on-windows-fe5332b0ca95., Erişim Tarihi: 10.04.2022.

[74] "Free Stock Photos, Royalty Free Stock Images & Copyright Free Pictures · Pexels." https://www.pexels.com/., Erişim Tarihi: 16.04.2022.

[75] “Welcome To Colaboratory- Colaboratory.” https://colab.research.google.com/., Erişim Tarihi: 16.04.2022.

ÖZGEÇMİŞ

Adı Soyadı **:** Nimetullah NECMETTİN

ÖĞRENİM DURUMU

Derece	**Eğitim Birimi**	**Mezuniyet Yılı**
Yüksek Lisans	Sakarya Üniversitesi / Fen Bilimleri Enstitüsü/Bilgisayar Mühendisliği	Devam ediyor
Yüksek Lisans	University of South-Eastern Norway / Computer Science	2021
Lisans	Oslo Metropolitan University / Computer Engineering	2020
Lisans	Xinjiang University (Doğu Türkistan)/Office Automation	2006
Lise	Korla (Doğu Türkistan) 1. Lisesi	2002

YABANCI DİL

Türkçe, İngilizce, Çince, Norveççe