T.C.
İNÖNÜ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME TABANLI GERÇEK ZAMANLI KİMLİKLENDİRME SİSTEMİ

YÜKSEK LİSANS TEZİ

Mehmet Fatih ÖZDEMİR

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Davut HANBAY

ARALIK - 2021

T.C
İNÖNÜ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

DERİN ÖĞRENME TABANLI GERÇEK ZAMANLI KİMLİKLENDİRME SİSTEMİ

YÜKSEK LİSANS TEZİ

Mehmet Fatih ÖZDEMİR
(36193619016)

Bilgisayar Mühendisliği Anabilim Dalı

Tez Danışmanı: Prof. Dr. Davut HANBAY

ARALIK - 2021

TEŞEKKÜR VE ÖNSÖZ

Bu tez çalışmasının her aşamasında yardım, öneri, bilgi, tecrübe ve desteklerini esirgemeden beni her konuda yönlendiren danışman hocam Prof. Dr. Davut HANBAY'a,

Tüm hayatım boyunca olduğu gibi tez çalışmalarım süresince de benden her türlü desteklerini esirgemeyen anneme, babama, eşime ve kızıma,

Tezin uygulama aşamasında "FYL-2021-2449" nolu proje numarasıyla vermiş oldukları maddi desteklerden dolayı, İnönü Üniversitesi BAP birimine

teşekkür ederim.

ONUR SÖZÜ

Yüksek lisans tezi olarak sunduğum "Derin Öğrenme Tabanlı Geçek Zamanlı Kimliklendirme Sistemi" başlıklı bu çalışmanın bilimsel ahlak ve geleneklere aykırı düşecek bir yardıma başvurmaksızın tarafımdan yazıldığına ve yararlandığım bütün kaynakların hem metin içinde hem de kaynakçada yöntemine uygun biçimde gösterilenlerden oluştuğunu belirtir, bunu onurumla doğrularım.

Mehmet Fatih ÖZDEMİR

İÇİNDEKİLER

ÇİZELGELER DİZİNİ

ŞEKİLLER DİZİNİ

SEMBOLLER VE KISALTMALAR

SGİ : Stokastik Gradyan İnişi
Gİ : Gradyan İnişi
DVM : Destek Vektör Makinesi
BTA : Bölge Teklif Ağı
OH : Ortalama Hassasiyet
R-CNN : Region Based Convolutional Neural Network
YOLO : You Only Look Once
FPS : Frame Per Second
mAP : Mean Average Precision
FPN : Feature Pyramid Network
IOU : Intersection over union
TP : True Positive
TN : True Negative
FP : False Positive
FN : False Negative
IDP : Identification precision
IDR : Identification Recall
IDF1 : Identification F1 Score
IDTP : Identification True Positive
IDFP : Identification False Positive
IDFN : Identification False Negative
TPA : True Positive Association
TNA : True Negative Association
FNA : False Negative Association
FPA : False Positive Association
MOT : Multi Object Tracking
MOTA : Multi Object Tracking Accuracy
HOTA : Higher Order Tracking Accuracy
LFW : Labeled Face in the Wild
YTF : Youtube Faces
LMCL : Large Margin Cosine Loss
API : Application Programming Interface
HTTP : Hyper-Text Transfer Protocol
TCP : Transmission Control Protocol

P-Net : Proposal Network

R-Net : Refine Network

O-Net : Output Network

NMS : Non-maximum Suppression

LMCL : Large Margin Cosine Loss

$\boldsymbol{b_t}$: t anındaki bulunan eşleşme sayısı

$\boldsymbol{h_t^i}$: t anındaki nesne-hipotez toplam uzaklığı

$\boldsymbol{n_t}$: t anındaki eksik tespit edilen sayısını

$\boldsymbol{yp_t}$: t anındaki yanlış pozitiflerin sayısını

$\boldsymbol{nne_t}$: t anındaki eşleşmeyenlerin sayısı

$\boldsymbol{\alpha}$: Yerelleştirme eşik değeri

$\boldsymbol{m}$: Marjin değeri

$\boldsymbol{k}$: Açısal kenar boşluğu

$\boldsymbol{W}$: Ağırlık vektörü

$\boldsymbol{B}$: Yanlılık vektörü

$\boldsymbol{\theta}$: W ile x arasındaki açı

ÖZET

Yüksek Lisans Tezi

Derin Öğrenme Tabanlı Geçek Zamanlı Kimliklendirme Sistemi

Mehmet Fatih ÖZDEMİR

İnönü Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı

62+VII sayfa

2021

Danışman: Prof. Dr. Davut HANBAY

Günümüzde nesne tespiti ve takibi en çok çalışılan alanlardan birisi olmuştur. Bunun sebebi günlük hayatta karşılaşılan güvenlik, savunma, medikal, robotik ve oto pilot araç kullanımlarında kritik öneme sahip olmasıdır. Bu amaçla yapay zekâ ve makine öğrenmesi kullanan birçok karar destek sistemi veya uzman sistem geliştirilmeye çalışılmıştır. Son zamanlarda derin öğrenme ve donanım alanında yaşanan gelişmelere bağlı olarak etkin ve güvenilir birçok nesne tespiti ve takip sistemi geliştirilmiştir.

Derin öğrenme algoritmalarının performansını arttırmak ve daha ekonomik sistemler geliştirmek amacıyla nesne tespiti ve takibi alanında çalışmalar hızla devam etmektedir. Yeni algoritmalar tasarlayan bilim adamları, hesaplama yükünü azaltacak ve performansı arttıracak modeller geliştirmek için çalışmalar yürütmektedir. Geliştirilen yeni modeller gerçek zamanlı uygulamalarda kullanılabilecektir.

Bu tez çalışmasının amacı derin öğrenme yaklaşımları kullanılarak çevrimiçi nesne takip sistemlerinin başarımını arttırmak ve günlük hayatta kullanılabilecek bir uygulama geliştirmektir. Tez, iki kısımdan oluşmaktadır. İlk olarak, çoklu nesne takibi derin öğrenme modeli FairMOT üzerinde mevcut optimizasyon algoritmalarının karşılaştırılması yapılmıştır. Hiper parametreler arasında yer alan optimizasyon algoritmaları mevcut yaklaşımlar denenerek daha başarılı bir model elde edilmiştir. MOT20 veri seti kullanılarak FairMOT derin öğrenme modeline farklı optimizasyon yöntemleri uygulanmıştır. En iyi sonuç RMSprop optimizasyon algoritması kullanılarak elde edilmiştir.

İkinci kısımda ise çoklu kamera sistemlerinde çalışan derin öğrenme tabanlı gerçek zamanlı bir yüz takip sistemi geliştirilmiştir. Yüz tespiti amacıyla SCRFD modeli, yüz tanıma için ise ArcFace modeli kullanılmıştır. Nesne takibi için ise yüz tespiti ve yüz tanıma modellerinden faydalanılarak DeepSORT algoritması kullanılmıştır. Gerçek zamanlı olarak verinin işlenebilmesi için Apache Kafka akış işleme sistemi ve Socket.IO çift yönlü iletişim kütüphanesinden yararlanılmıştır.

Yapılan çalışma sonucunda başarılı bir yüz tespit, tanıma ve takip sistemi uygulaması geliştirilmiştir.

Anahtar Kelimeler: Derin Öğrenme, Çoklu Nesne Takibi, Çoklu Kamera, Yüz Tespiti, Yüz Tanıma, Yüz Takibi

ABSTRACT

Master Thesis

Deep Learning Based Real Time Identification System

Mehmet Fatih ÖZDEMİR

Inonu University
Graduate School of Nature and Applied Sciences
Department of Computer Engineering

62+VII pages

2021

Supervisor: Prof. Dr. Davut HANBAY

Nowadays, object detection and tracking has become one of the most studied areas. The reason for this is that it is of critical importance in the use of security, defense, medical, robotic and autopilot vehicles encountered in daily life. For this purpose, many decision support systems or expert systems using artificial intelligence and machine learning have been tried to be developed. Recently, depending on the developments in the field of deep learning and hardware, many effective and reliable object detection and tracking systems have been developed.

In order to increase the performance of deep learning algorithms and to develop more economical systems, studies in the field of object detection and tracking continue rapidly. Scientists who design new algorithms are working to develop models that will reduce the computational cost and increase performance. Developed new models can be used in real-time applications.

The aim of this thesis is to increase the performance of online object tracking systems by using deep learning approaches and to develop an application that can be used in daily life. The thesis consists of two parts. First, a comparison of existing optimization algorithms on the multi-object tracking deep learning model FairMOT is made. Optimization algorithms, which are among the hyper-parameters, have been tried and a more successful model has been obtained. Different optimization methods were applied to the FairMOT deep learning model using the MOT20 dataset. The best results were obtained using the RMSprop optimization algorithm.

In the second part, a deep learning based real-time face tracking system working on multiple camera systems has been developed. SCRFD model was used for face detection and ArcFace model was used for face detection. For object tracking, the DeepSORT algorithm was used by making use of face detection and face recognition models. Apache Kafka stream processing system and Socket.IO bidirectional communication library were used to process data in real time.

As a result of the study, a successful face detection, recognition and tracking system application has been developed.

Keywords: Deep Learning, Multi Object Tracking, Multi Camera, Face Detection, Face Recognition, Face Tracking

1. GİRİŞ

Yapay zekânın gelişmesiyle birlikte birçok yenilik hayatımıza girmiştir. Geleneksel sinyal ve görüntü işleme yöntemlerinin literatüre katkısı yadsınamaz derecede büyüktür. Sinyal ve görüntü işlemede yapay zekâ çözümleri ortaya çıkana kadar birçok bilimsel çalışmanın ana kaynağı geleneksel yöntemlerdi. Gelişen donanım ve yapay zekâ yaklaşımları ile birlikte daha verimli ve başarılı sonuçlar elde edilmiştir. Yapay zekânın daha verimli ve başarılı kazanımları ile birlikte kullanım oranı artmaya başlamıştır. Yapay zekâ birçok alt bilim dalına sahiptir. En çok kullanılanlardan birisi şüphesiz Makine öğrenmesidir. Makine öğrenmesi, makinelerin insanlar gibi veri analizi yaparak öğrenmelerini sağlayan yapay zekânın alt bilim dalıdır.

Makine öğrenmesinde denetimli öğrenme, denetimsiz öğrenme ve pekiştirmeli öğrenme olmak üzere üç yaklaşım bulunmaktadır. Denetimli öğrenme, etiketlenmiş veriler üzerinde algoritmalar öğrenmeyi gerçekleştirir [1]–[3]. Bu yaklaşımda etiketlemiş verilerin dışında bir sonuç beklenilemez. Etiketli nesneler üzerinde sınıflandırma işlemi denetimli öğrenmeye bir örnektir. Denetimsiz öğrenmede ise veriler etiketlenmemiştir. Veriler kendi aralarındaki benzerlik derecesine göre etiketlemektedir. Bu yaklaşım, bilinmeyen durumlar için faydalıdır. Örneğin, müşteri verileri üzerinden benzer ürünleri alan müşterilerin ürün bazlı kümelemelerinin oluşturulması işlemi denetimsiz öğrenmedir. Son olarak pekiştirmeli öğrenmede ise her olaydan veya durumdan sonra ortaya çıkan sonucun doğruluk durumunu belirlemeye yardımcı olan geri bildirimlerden faydalanılmaktadır. Otomatikleştirilmiş sistemler için kullanılabilecek iyi bir tekniktir. Örneğin, sürücüsüz araçlar pekiştirme geçmişi kazandıkça hız limitini aşmamayı, şeritte kalmayı ve yayaları fark edince fren yapmayı öğrenir.

Derin öğrenme, yapay sinir ağlarını temel alan bir veya daha fazla gizli katman içeren makine öğrenme algoritmalarını kapsayan bir bilim alanıdır. Bu yaklaşımda her katman, giriş verilerini bir sonraki katmanın belirli bir görev için kullanabileceği giriş bilgilerine dönüştüren birimlerden oluşur. Bu yapı sayesinde algoritma kendi veri işleme süreçleriyle bilgi sahibi olmaktadır. Derin öğrenme modelleri, veri süreçleri hakkında bilgi sahibi

olurken optimizasyon algoritması, öğrenme katsayısı vb. hiper parametrelerin doğru seçilmesi başarıyı ciddi anlamda etkilemektedir [4]. Derin öğrenmede birçok alt çalışma alanı vardır. Nesne tespiti, nesne takibi, yüz tespiti, yüz tanıma bunlardan bazılarıdır. Nesne tespiti, sınıflandırma ve sınırlayıcı kutuları öğrenmeye çalışırken, nesne takibi bir sonraki adım olarak ilişkilendirmeyi öğrenmeye çalışır [1]–[3], [5]. Derin öğrenme modellerinin bazılarında nesnenin tespiti ve takibi tek model üzerinde gerçekleşebileceği gibi ayrı modellerde de ele alınarak da yapılabilmektedir [6]–[9]. Diğer taraftan genel nesne tespitinden farklı olarak yüz tespiti, insan yüzüne özgü öznitelikleri çıkararak görüntü üzerindeki yüzlerin konum tespitlerinin yapılması amaçlanmaktadır [10]–[12]. Yüz tanıma ise insanlar arası sınıflandırma probleminden başka bir şey değildir [13]–[15]. Yüz tanımada açık küme ve kapalı küme olmak üzere iki yaklaşım benimsenmiştir. Açık küme yaklaşımında sınıflandırma, derin öğrenme modeli ile yüze ait öznitelikleri çıkarıldıktan sonra en yakın komşu metriği ile karşılaştırma yapılmaktadır. Kapalı küme yaklaşımında sınıflandırma işlemi için direk eğitilen derin öğrenme modeli kullanılmaktadır. Bunun için sınıflandırmaya dahil edilecek kişiler önceden eğitim veri setinde yer almak zorundadır.

Çoklu kamera kullanılan sistemlerde gerçek zamanlı olarak yüz tespiti ve yüz tanımlama işlemlerinin uygulanması kolay bir işlem değildir. Bunun için her kameradaki akış görüntüleri birbirinden bağımsız olarak ayrı ayrı işlenmelidir. Yani, kameraların eş zamanlı olarak ele alınması gerekmektedir. Ayrıca kamera akışından gelen yeni görüntü önceki görüntü ile ilişkisiz olarak değerlendirilerek yüz tespit ve yüz tanımlama işlemi yapılabilmektedir. Bu durum önceki görüntülerdeki bilgilerin kullanılmasının önüne geçmektedir. DeepSORT takip algoritması ile yüz tanıma işleminin her görüntüde tekrarlanmasının önüne geçilerek daha kararlı ve daha düşük maliyetli bir sistem gerçekleştirilmiştir. Ayrıca sistemin eş zamanlı olarak yorumlanabilmesi, veri kaybının önlenebilmesi ve akış görüntülerinin gerçek zamanlı işlenerek görüntülenebilmesi amacıyla Apache Kafka olay akış platformu [16] ve Socket.IO [17] kütüphanesi kullanılmıştır.

1.1. Tezin Amacı

Bu tez çalışmasında derin öğrenme ile çevrimiçi nesne takibinin yapılması amaçlanmıştır. Tezde uygulamalar bölümünde yer verilen çalışmalar ile ilk olarak literatüre katkı sağlanması hedeflenmektedir. İlk olarak gerçek zamanlı nesne takibi için kullanılan FairMOT modeline hiper parametre optimizasyon algoritmalarının etkisi incelenmiştir.

Daha sonra yapılan gerçek zamanlı derin öğrenme tabanlı yüz tespiti tanıması ve takibi uygulaması gerçekleştirilmiştir.

1.2. Tezin Organizasyon Yapısı

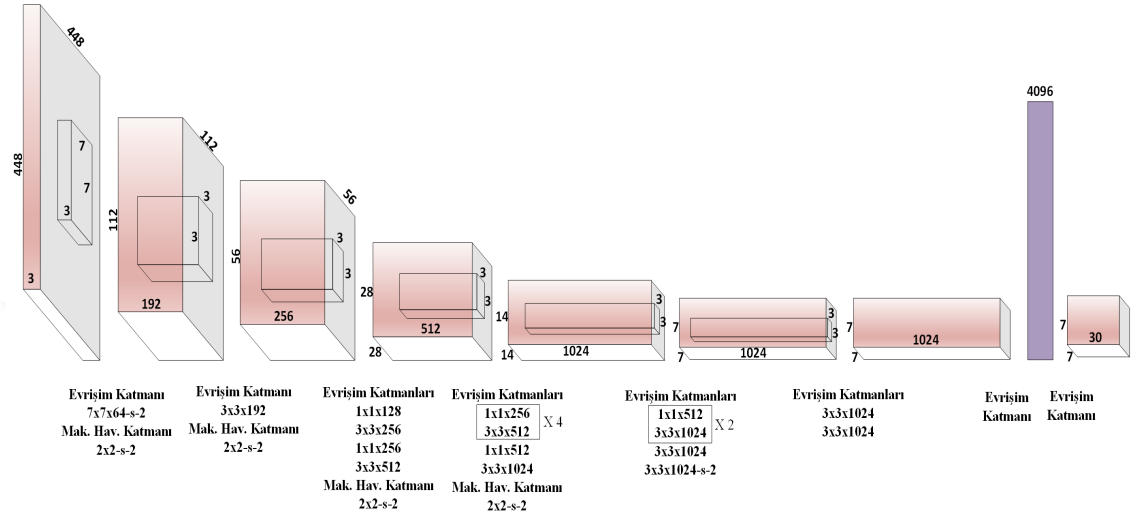
Bölüm 2'de nesne tespiti alanında yapılan çalışmalara yer verilmiştir. Bölüm 3'te nesne takibi ve derin öğrenme modelleri alanında gerçekleştirilen çalışmalara yer verilmiştir. Bölüm 4'te yüz tespiti alanından yapılan güncel çalışmalar incelenmiştir. Bölüm 5'te yüz tanıma alanında yapılan güncel çalışmalar incelenmiştir. Bölüm 6'da Apache Kafka dağıtık olay akış platformundan açıklanmıştır. Bölüm 7'de çift yönlü ve olay tabanlı iletişim sağlayan Socket.IO kütüphanesi açıklanmıştır. Bölüm 8'de tez konusu kapsamında yapılan çalışmalara ve uygulamalara yer verilmiştir. Bölüm 9'da ise tezde yapılan çalışmalara ait sonuçlar ve gelecekte yapılan çalışmalara yön vermek için öneriler sunulmuştur.

2. NESNE TESPİTİ

Nesne tespiti, görüntüler üzerinde nesnelerin tespitini amaçlayan, gelişen bilgisayar donanımları ile birlikte yükseliş eğiliminde olan çalışma alanlarından biridir. Nesne tespiti nesne sınırlarının belirlenmesi ve sınıf olasılıklarının hesaplanması olmak üzeri genel olarak iki aşamadan oluşur [1], [2]. Nesne tespitinin başarısı bu iki aşamanın başarısına bağlıdır. Bu iki aşamayı tek bir ağ ve tek bir değerlendirmede çözen modellerin yanı sıra iki ayrı ağda çözüm üreten modeller mevcuttur. Tekli model ile çözen YOLO [1] ile birlikte iki aşama da çözen R-CNN [3], Fast R-CNN [2], Faster R-CNN [5] gibi modellere literatürde yer verilmiştir. Bu iki yaklaşım arasındaki genel farklılık, ikili modellerde başarı daha yüksek olurken tekli modeller gerçek zamanlı sistemlere daha uygundur.

YOLO tek bir sinir ağı ve tek bir değerlendirmede doğrudan görüntüdeki sınırlayıcı kutuları ve sınıf olasılıklarını tahmin eder. Nesne tespitinin tamamı tek bir ağ olduğundan, uçtan uca doğrudan nesne tespit performansına göre optimize edilebilir. Birleşik mimari son derece hızlıdır. Önerildiği yıldaki teknoloji ürünü algılama sistemleriyle karşılaştırıldığında, YOLO daha fazla yerelleştirme hatası yapar ancak arka planda yanlış pozitifleri tahmin etme olasılığı daha düşüktür. R-CNN [3] gibi yaklaşımlar, önce bir görüntüde potansiyel sınırlayıcı kutular oluşturmak ve ardından önerilen bu kutular üzerinde bir sınıflandırıcı çalıştırmak için bölge önerme yöntemlerini kullanır. Sınıflandırmadan sonra, sınırlayıcı kutuları iyileştirmek, çift algılamaları ortadan kaldırmak ve sahnedeki diğer nesnelere dayalı olarak kutuları yeniden puanlamak için ek bir işlem daha yapmaktadır. R-CNN'deki bu karmaşık işlem hatları yavaştır ve optimize edilmesi zordur çünkü her bir bileşenin ayrı ayrı eğitilmesi gerekir. YOLO, nesne algılamayı, doğrudan görüntü piksellerinden sınırlayıcı kutu koordinatlarından sınıf olasılıklarına kadar tek bir regresyon problemi olarak yeniden ele almıştır. YOLO, hangi nesnelerin olduğunu ve nerede olduklarını tahmin etmek için bir görüntüye yalnızca bir kez bakmaktadır. Kayan pencere ve bölge önerisi tabanlı tekniklerin aksine, YOLO eğitim ve test süresi boyunca görüntünün tamamını görür, böylece sınıflar ve görünümleri hakkındaki bağlamsal bilgileri model içerisinde kodlar. Fast R-CNN [2], daha geniş bağlamı göremediği için bir görüntüdeki arka plan yamalarını nesnelerle karıştırır. YOLO, Fast R-CNN'ne [2], kıyasla arka plan hata sayısının yarısından daha azını yapar. Şekil 2.1'de mimarisi verilen ağın ilk evrişimsel katmanları, görüntüden özellikleri çıkarırken, tamamen bağlı katmanlar çıktı olasılıklarını ve koordinatları tahmin eder. Ağın mimarisi, görüntü sınıflandırması için GoogLeNet [18] modelinden ilham alınarak

tasarlanmıştır. Ağ mimarisinde 24 evrişim katmanı ve ardından 2 tam bağlantı katmanına yer verilmiştir. Ağın son çıkış katmanı olarak 7x7x30 bir tensördür. Aynı çalışma altında hızlı versiyon olarak Fast YOLO mimarisi önerilmiştir. Fast YOLO versiyonunda ise eğitim ve test parametreleri aynı olmasına rağmen 24 evrişim katmanı yerine 9 evrişim katmanı kullanılmıştır [1].



Şekil 0.1 : YOLO mimarisi [1].

YOLO, her ızgara hücresi için yalnızca iki kutu öngördüğü ve yalnızca bir sınıfa sahip olabileceği için sınırlayıcı kutu tahminlerine güçlü uzamsal kısıtlamalar getirir. Bu uzamsal kısıtlama, modelin tahmin edebilebileceği yakındaki nesnelerin sayısını sınırlar. Kuş sürüleri gibi gruplar halinde görünen küçük nesnelerle mücadele eder. Verilerden sınırlayıcı kutuları tahmin etmeyi öğrendiğinden, yeni veya olağandışı en boy oranlarındaki veya konfigürasyonlarındaki nesnelere genelleme yapmakta zorlanır. Son olarak, algılama performansına yaklaşan bir kayıp fonksiyonu üzerinde eğitim yaparken, kayıp fonksiyonu küçük sınırlayıcı kutulardaki hatalara büyük sınırlayıcı kutulardaki gibi davranır. Büyük bir kutudaki küçük bir hata genellikle zararsızdır, ancak küçük bir kutudaki küçük bir hata, birleşim üzerindeki kesişimde (Intersection over Union (IOU)) çok daha büyük bir etkiye sahiptir. Modelde ana hata kaynağı genellikle yanlış yerelleştirmeler olarak görünmektedir [1].

YOLO9000, 9000'den fazla nesne kategorisini algılayabilen gerçek zamanlı nesne algılama sistemi olarak karşımıza 2017 yılında çıkmıştır. Önceki çalışmalardan alınan YOLO [1] algılama yönteminde çeşitli iyileştirmeler ve yenilikler önerilmiştir. Çok ölçekli

bir eğitim yöntemi kullanan yeni YOLOv2 modeli, hız ve doğruluk arasında kolay bir uyum sunarak değişen boyutlarda çalışabilmektedir. YOLOv2, 67 (Frame per second(FPS))'de, VOC 2007 [19] veri setinde 76.8 ((mean average precision)mAP) değerine ulaşırken 40 FPS'de YOLOv2 78.6 mAP değerini güncellemektedir ve Faster R-CNN [5] gibi yöntemlerden daha iyi performans gösterirken aynı zamanda önemli ölçüde daha hızlı çalışır. Nesne algılama ve sınıflandırma konusunda ortak eğitim için yeni bir yöntem ile YOLO9000 modeli aynı anda COCO [20] algılama veri seti ve ImageNet [21] sınıflandırma veri seti üzerinde eğitilmiştir. Ortak eğitim sayesinde, YOLO9000 modeli etiketli algılama verilerine sahip olmayan nesne sınıfları için de tahmin yapılmasına olanak sağlar. YOLO9000, 200 sınıfın yalnızca 44'ü için algılama verisine sahip olmasına rağmen ImageNet doğrulama setinde 19.7 mAP değerine sahiptir. COCO veri setinde var olmayan 156 sınıf için YOLO9000 modeli 16.0 mAP değerine sahiptir. YOLO9000 ile birlikte büyük miktarda sınıflandırma verisini kullanmak için yeni bir yöntem önerilerek mevcut algılama sistemlerinin kapsamı genişletilmiştir. İlk olarak, gerçek zamanlı bir dedektör olan YOLOv2'yi üretmek için temel YOLO [1] algılama sistemi güncellenmiştir. Ardından, 9000'den fazla farklı nesne kategorisini algılayabilen gerçek zamanlı bir nesne dedektörü olan YOLO9000 modelinin üretilmesi için ImageNet'te 9000'den fazla sınıf ile COCO'daki algılama verileri kullanılmıştır. Modeli eğitmek için veri kümesi kombinasyon yöntemi ve ortak eğitim algoritması kullanılmıştır. Ayrıca, YOLOv2 ile ağı büyütmek yerine ağı basitleştirerek öğrenme kolaylaştırılmıştır. YOLO'daki [1] tüm evrişim katmanlarına toplu normalleştirme eklenerek mAP'de %2'den fazla iyileştirme elde edilmiştir. YOLO [1] modeli, sınıflandırıcı ağı 224 × 224 girdi boyutuyla eğitir ve algılama için çözünürlüğü 448'e yükseltir. Bu da ağın aynı anda öğrenebilmesi için yeni giriş çözünürlüğüne uyum sağlaması gerektiği anlamına gelir. YOLOv2 için önce ImageNet'te 10 tur 448×448 boyutunda çözünürlük ile sınıflandırma ağına ince ayar yapılır. Bu yaklaşım, ağ filtrelerinin daha yüksek çözünürlüklü giriş ile daha iyi çalışacak şekilde ayarlaması için zemin hazırlar. Ardından, son olarak ortaya çıkan ağa algılamada ince ayar yapılır. Mevcut ayarlama ile yüksek çözünürlüklü sınıflandırma ağı üzerinde hemen hemen %4 mAP değerinde artış sağlanmıştır. Tamamen bağlı katmanlar, YOLO'dan kaldırılmıştır ve sınırlayıcı kutuları tahmin etmek için çapa kutuları kullanılmaktadır. Ayrıca, YOLOv2'nin temeli olarak kullanılacak yeni bir sınıflandırma modeli olan ve Çizelge 2.1'de mimarisi verilen Darknet-19 ağı önerilmiştir [6].

Çizelge 0.1 : Darknet-19 mimarisi [6].

Tür	Filtreler	Boyut/Adım	Çıktı
Evrişim	32	3 x 3	224 x 224
Mak.Havuzlama		2 x 2/2	112 x 112
Evrişim	64	3 x 3	112 x 112
Mak. Havuzlama		2 x 2/2	56 x 56
Evrişim	128	3 × 3	56 × 56
Evrişim	64	1 x 1	56 x 56
Evrişim	128	3 × 3	56 × 56
Mak. Havuzlama		2 x 2/2	28 x 28
Evrişim	256	3 × 3	28 x 28
Evrişim	128	1 × 1	28 x 28
Evrişim	256	3 × 3	28 x 28
Mak. Havuzlama		2 x 2/2	14 x 14
Evrişim	512	3 × 3	14 x 14
Evrişim	256	1 × 1	14 x 14
Evrişim	512	3 × 3	14 x 14
Evrişim	256	1 × 1	14 x 14
Evrişim	512	3 × 3	14 x 14
Mak. Havuzlama		2 x 2/2	7 x 7
Evrişim	1024	3 × 3	7 x 7
Evrişim	512	1 × 1	7 x 7
Evrişim	1024	3 × 3	7 x 7
Evrişim	512	1 × 1	7 x 7
Evrişim	1024	3 × 3	7 x 7
Evrişim	1000	1 x 1	7 x 7
Ort. Havuzlama		Genel	1000
Softmax			

YOLOv3 [7], YOLOv2'nin gelişmiş versiyonu olarak ortaya çıkmıştır ve bir takım iyileştirmeler yapılmıştır. Yeni bir sınıflandırma modeli olan Darknet-53, YOLOv3'de omurga ağı olarak kullanılmıştır. Darknet-53 mimarisi Çizelge 2.2'de verilmiştir. YOLOv3, lojistik regresyon kullanarak her sınırlayıcı kutu için bir nesnellik puanı tahmin eder. İyi performans almak adına gereksiz olduğunu düşünüldüğü için Softmax yerine sadece bağımsız lojistik sınıflandırıcılar kullanır. Softmax kullanmak, her kutunun tam olarak bir sınıfa sahip olduğu varsayımını uygular fakat genellikle durum böyle değildir. Her kutu, çok etiketli sınıflandırmayı kullanarak sınırlayıcı kutunun içerebileceği sınıfları tahmin eder. Çok etiketli sınıflandırma yaklaşımının, verileri daha iyi modellediği tespit edilmiştir. Eğitim sırasında sınıf tahminleri için çapraz entropi kullanılmaktadır. Çok ölçekli tahminlerle YOLOv3'ün nispeten yüksek ortalama hassasiyet (AP) performansına sahip olduğu gözlemlenmiştir. Bununla birlikte, orta ve daha büyük boyutlu nesnelerde nispeten daha kötü bir performansa sahiptir [7].

Çizelge 0.2 : Darknet-53 mimarisi [7].

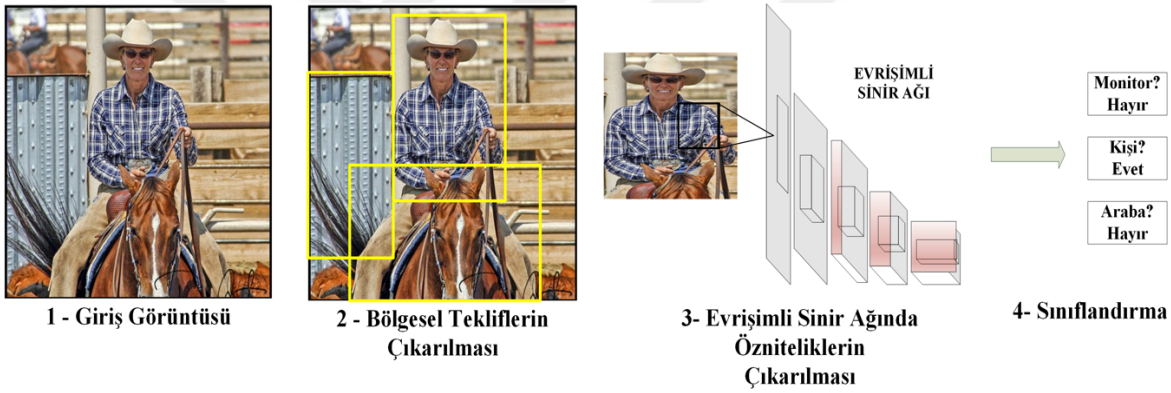
	Tür	Filtreler	Boyut/Adım	Çıktı
	Evrişim	32	3 x 3	256 x 256
	Evrişim	64	3 x 3/2	128 x 128
	Evrişim	32	1 x 1	
1 x	Evrişim	64	3 x 3	
	Atık			128 x 128
	Evrişim	128	3 x 3/2	64 x 64
	Evrişim	64	1 x 1	
2 x	Evrişim	128	3 x 3	
	Atık			64 x 64
	Evrişim	256	3 × 3/2	32 x 32
	Evrişim	128	1 x 1	
8 x	Evrişim	256	3 x 3	
	Atık			32 x 32
	Evrişim	512	3 × 3/2	16 x 16
	Evrişim	256	1 x 1	
8 x	Evrişim	512	3 x 3	
	Atık			16 x 16
	Evrişim	1024	3 × 3/2	8 x 8
	Evrişim	512	1 x 1	
4 x	Evrişim	1024	3 x 3	
	Atık			8 x 8
	Ort. Havuzlama		Genel	
	Tam Bağlantı		1000	
	Softmax			

YOLOv4, YOLOv3'ün ortalama hassasiyetini (AP) %10'a kadar ve FPS değerini ise %12'ye kadar arttırmıştır. YOLOv4'de omurga ağı olarak CSPDarknet53 kullanılmıştır. CSPDarknet53, DarkNet53 ağını kullanan nesne tespiti için evrişimli bir sinir ağı omurgasıdır. CSPDarknet53'de temel katmanın özellik haritasını iki parçaya bölmek için CSPNet [22] yaklaşımı kullanılır ve parçaları aşamalar arası bir hiyerarşi aracılığıyla birleştirir. Böl ve birleştir stratejisinin kullanılması, ağ üzerinden daha fazla gradyan akışına izin vermektedir [8].

Mevcut derin evrişimli sinir ağları giriş olarak sabit boyutlu (örneğin 32x32) bir giriş görüntüsü gerektirir. Bu gereklilik rastgele bir boyuttaki/ölçekteki görüntüler veya alt görüntüler için tanıma doğruluğunu azaltabilir. Sabit boyut gereksinimi ortadan kaldırmak için modelde havuzlama stratejisi olan "uzamsal piramit havuzlama" [23] adında yeni bir teknik kullanılmaktadır. CSPDarknet53 ağında da aynı şekilde uzamsal piramit havuzlama bloğu kullanılarak duyarlı alanı önemli ölçüde arttırarak en önemli bağlam özelliklerini

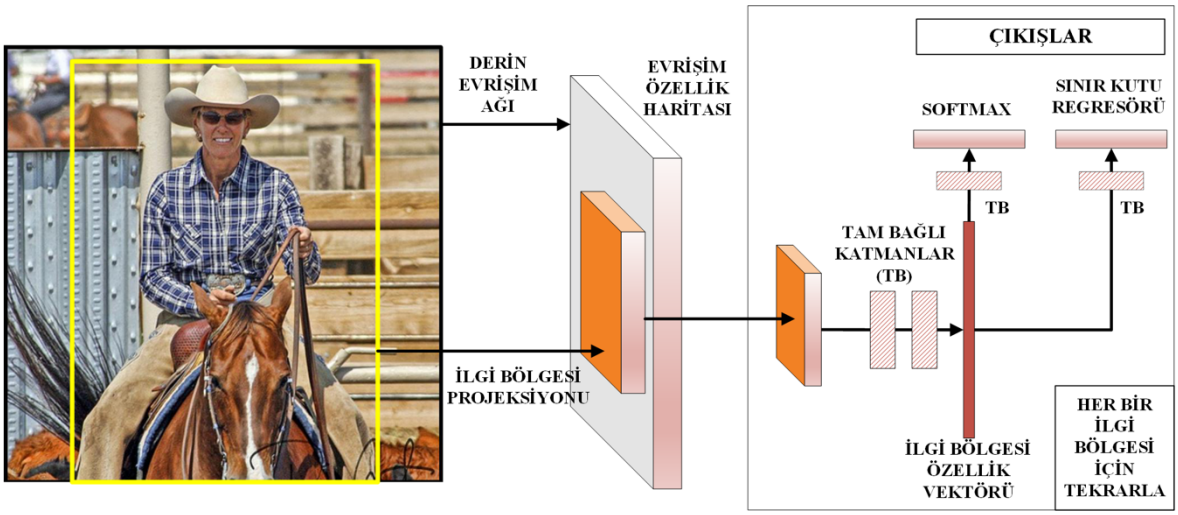
çıkarmaktadır. Ağın çalışma hızında neredeyse hiç azalmaya neden olmamaktadır. YOLOv3'te kullanılan (Feature Pyramid Network(FPN)) [24] yerine, farklı detektör seviyeleri için farklı omurga seviyelerinden öznitelik toplama yöntemi olarak PANet [25] ağı kullanılmaktadır [8].

R-CNN [3], ilk olarak nesneleri yerelleştirmek ve bölütlere ayırmak için aşağıdan yukarıya bölge tekliflerine yüksek kapasiteli evrişimli sinir ağları uygular. Şekil 2.2'de gösterildiği gibi model üç modülden oluşmaktadır. Birinci modülde, kategoriden bağımsız bölge teklifleri üretilir. İkinci modülde, her bölgeden sabit uzunlukta bir öznitelik vektörü çıkarmak için büyük bir evrişimli sinir ağı kullanılır. Evrişimli sinir ağı 227x227 boyutunda görüntüleri girdi olarak almaktadır ve her bölge için 4096 boyutunda özellik vektörü çıkarılmaktadır. Üçüncü modülde ise, destek vektör makinesi (DVM) [26] kullanılarak çıkarılan öznitelikler sınıflandırılır.



Şekil 0.2 : R-CNN Derin öğrenme modeli [3].

Fast R-CNN [2], R-CNN [3] modeline göre eğitim ve test hızını iyileştirirken aynı zamanda algılama doğruluğunu artırmak için çeşitli yenilikleri kullanır. Mimari çizimi Şekil 2.3'te gösterilen Fast R-CNN, girdi olarak bir görüntü ve bir dizi nesne önerisi almaktadır. Ağ, bir evrişim özellik haritası oluşturmak için önce tüm görüntüyü birkaç evrişimli sinir ağında maksimum havuzlama katmanıyla işler. Ardından, her nesne teklifi için bir ilgi bölgesi havuzlama katmanı ve öznitelik haritasından sabit uzunlukta bir özellik vektörü çıkarır. Her özellik vektörü, sonunda iki çıktı katmanına ayrılan bir dizi tam bağlantılı katmanını besler. Bunlardan biri nesne sınıfları üzerinde Softmax fonksiyonu kullanılarak nesne olasılık tahminleri üretirken; diğeri ise nesne sınır kutu konumlarını çıkarır.



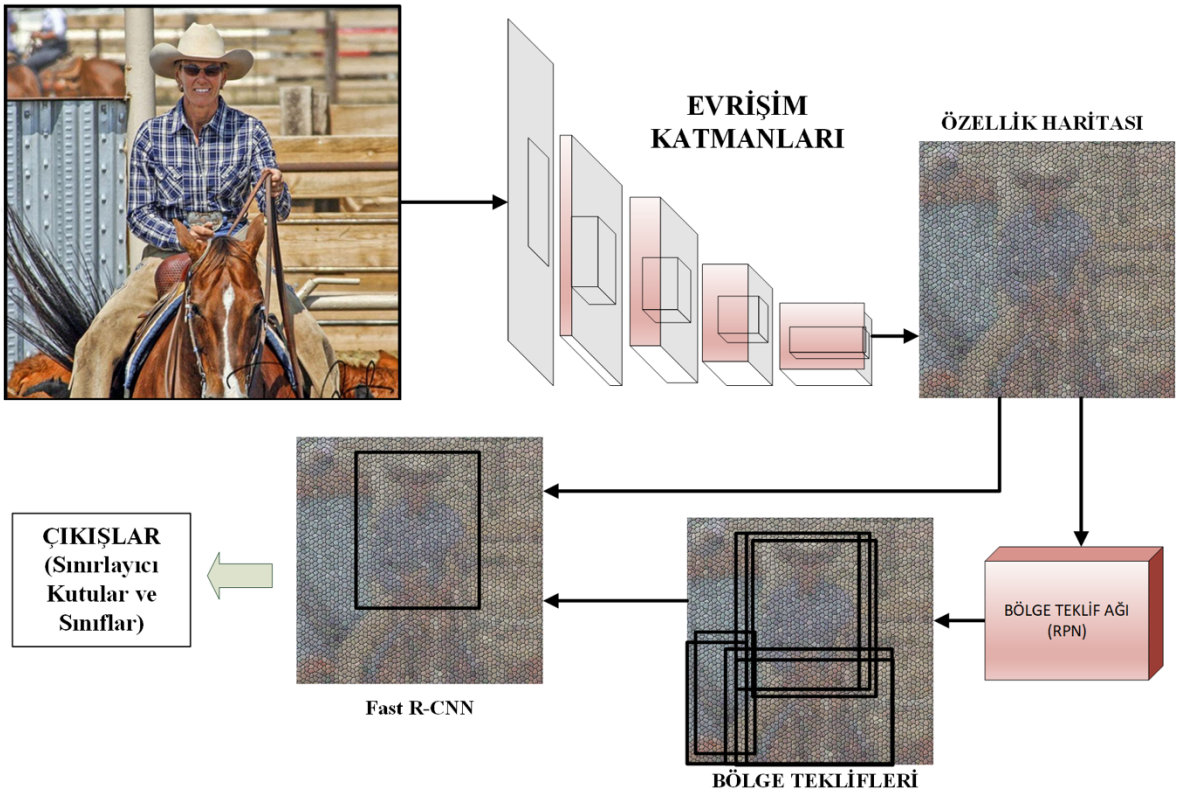
Şekil 0.3 : Fast R-CNN Derin öğrenme modeli [2].

Faster R-CNN [5], Fast R-CNN [2] algoritmasına ek olarak algılama ağı ile tam görüntü evrişim özelliklerini paylaşan ve bağımsız bölge önerileri sağlayan bir bölge teklif ağı (BTA) ekleyerek yeni iyileştirmeler yapmıştır. Faster R-CNN iki modülden oluşmaktadır. Şekil 2.4'te gösterildiği gibi ilk modül, bölgeleri teklif eden derin öğrenme modeli iken ikinci modül Fast R-CNN dedektörüdür.

Bölge teklif ağı (BTA), girdi olarak herhangi bir boyutta giriş görüntüsünü alır ve nesne skoruna göre sınırlayıcı kutu teklifi ortaya çıkarır. Bu mini ağ, kayar pencere şeklinde çalıştığından, tam bağlantılı katmanlar tüm uzamsal konumlar arasında paylaşılmaktadır. Bu mimari bir n x n evrişim katmanı ve ardından iki kardeş 1 x 1 evrişim katmanı ile uygulanır. Bölge teklifini, evrişimli katmanlar tarafından oluşturulan öznitelik haritası üzerinde küçük bir ağı kaydırarak yapar. Bölge teklif ağı, stokastik gradyan inişi [27] ile uçtan uca eğitilmiştir. Kayan pencere tekniği ile çıkarılan tüm çapaların kayıp fonksiyonları için optimize etmek mümkündür fakat fazlaca negatif örneklere sebep olacaktır. Bunun yerine, örneklenen pozitif ve negatif çapaların 1:1 oranına sahip olduğu en küçük yığın kayıp fonksiyonunu hesaplamak için bir görüntüdeki 256 çapa rastgele seçilir. Bir görüntüde 128'den az pozitif örnek varsa, küçük toptan negatif olanlarla doldurulur. Paylaşılan evrişim katmanları, ImageNet sınıflandırması için ön eğitimden geçirilir. Sonraki adımda model PASCAL VOC veri setinde ilk 60 bin görüntü için 0,001 ve sonraki 20 bin görüntü için 0,0001 öğrenme katsayısı ile eğitilmiştir.

İkinci modül, önerilen bölgeleri kullanan Fast R-CNN [2] detektörüdür. Bölge telif ağ (BTA) modülü, Fast R-CNN modülüne nereye bakacağını söyler.

Tüm sistem ilk etapta çıkarılan özellik haritasını, bölge teklif ağı ve Fast R-CNN dedektörü ile paylaşarak tek bir ağda birleştirir. Model, 5-17 FPS hızında çalışan birleşik, derin öğrenme tabanlı bir nesne tespiti sistemi sunar. Ayrıca önerilen bölge teklif ağı, bölge teklif kalitesini ve dolayısıyla genel nesne algılama doğruluğunu iyileştirir.



Şekil 0.4 : Faster R-CNN mimarisi [4].

3. NESNE TAKİBİ

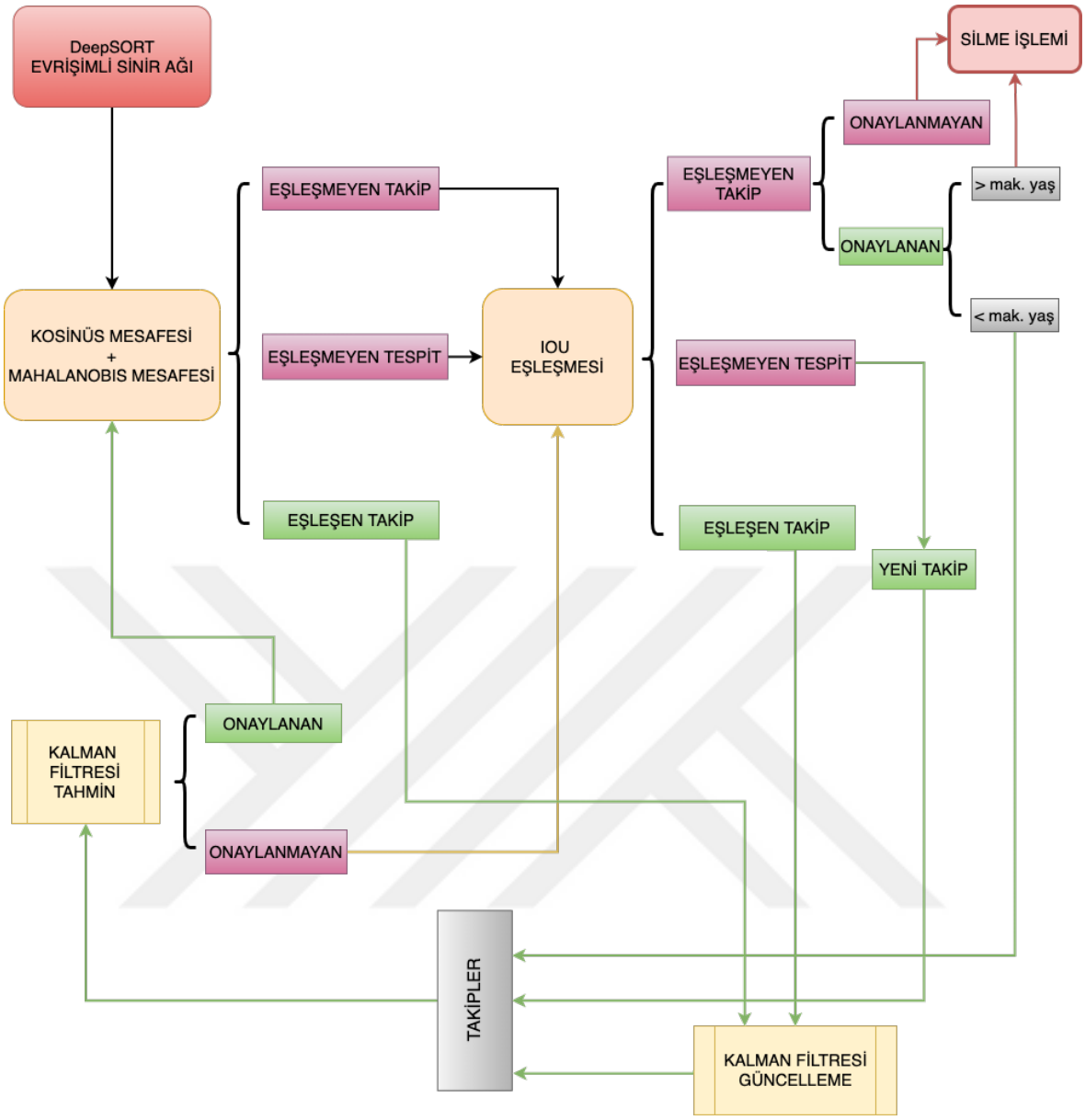
Nesne takibi, video görüntüleri üzerinde görüntü çerçevesinde tespit edilen nesnenin sonraki çerçevelerde ilişkilendirilerek takip edilmesi olayıdır. Çoklu nesne takibi ise nesne takibine ek olarak birden fazla nesnenin çerçeve üzerinde takibinin yapılması işlemidir. Nesne tespitini ve takibini tek bir model ile gerçekleştiren yaklaşımların CenterTrack [28], FairMOT [29], ByteTrack [30] yanı sıra sadece nesne takip görevini yerine getiren DeepSORT [9] gibi yaklaşımlarda mevcuttur. Nesne takibinde tespit aşamasına ek olarak ilişkilendirme aşaması da vardır. İlişkilendirme aşaması, nesne takibinin kapsamı içerisinde yer alan kişi takibi çalışma alanlarında kimliklendirme veya yeniden kimliklendirme olarak da tanımlanabilmektedir.

3.1. DeepSORT

DeepSORT [9] algoritması sadece nesneleri takip etmeyi kendine görev edinmiş bir algoritmadır. Bu anlamda hem nesne tespiti hemde nesne takibi yapan yaklaşımlardan ayrılmaktadır. DeepSORT, SORT [31] algoritmasının performasını arttırmak için görünüm bilgilerini yeni algoritmaya entegre ederek yeni bir yaklaşım sunmuştur. Deneysel değerlendirmeler sonucunda, yeni iyileştirmelerin kimlik değiştirme sayısını %45 oranında azalttığı ve yüksek FPS hızlarında genel rekabetçi performans elde edildiği ifade edilmiştir. SORT, izleme hassasiyeti ve doğruluğu açısından genel olarak iyi bir performans elde ederken, nispeten yüksek sayıda kimlik değişimine sebep olmaktadır. Bunun nedeni, kullanılan ilişkilendirme metriğinin, yalnızca durum tahmini belirsizliğinin düşük olduğu durumlar da doğru olmasıdır. Bu nedenle, SORT algoritması tıkanıklıkları takip etmede eksikliğe sahiptir. DeepSORT, ilişki metriğini hareket ve görünüm bilgileriyle birleştiren daha bilinçli bir metrikle değiştirerek bu sorunun üstesinden gelmektedir. Özellikle, büyük ölçekli kişi veri setinde yayaları ayırt etmek için eğitilmiş bir evrişimsel sinir ağı uygulanmaktadır. Bu ağın entegrasyonu sayesinde, sistemin uygulanmasını kolay, verimli ve çevrimiçi senaryolara uygulanabilir tutarken, ıskalamalara ve tıkanmalara karşı sağlamlığı arttırmaktadır.

Şekil 3.1'de akış şeması verilen DeepSORT algoritması, ilk olarak tespit edilen nesneleri kullanarak evrişimli sinir ağından öznitelikleri çıkartır. Çıkarılan öznitelikler eşleşen basamaklı bir dizi mesafe ölçüm algoritmalarından geçirilmektedir. Bu algoritmalar Mahalanobis ve Kosinüs mesafe ölçüm hesaplamalarıdır. Mahalanobis mesafesi, hareket

belirsizliğin düşük olduğu durumlarda daha uygun bir ilişkilendirme metriğidir. Fakat açıklanamayan kamera hareketleri görüntü düzleminde hızlı yer değiştirmelere neden olabilir, bu da Mahalanobis mesafesini tıkanıklıklar boyunca takip işlemini oldukça kullanışsız bir metrik haline getirir. Bundan dolayı ikinci bir metrik olarak Kosinüs mesafe ölçüm metriğinin eklenmesine ihtiyaç duyulmuştur. Mahalanobis mesafesi, özellikle kısa vadeli tahminler için yararlı olan harekete dayalı olası nesne konumları hakkında bilgi sağlarken Kosinüs mesafesi, hareketin daha az ayırt edici olduğu uzun süreli tıkanıklıklardan sonra kimlikleri geri kazanmak için özellikle yararlı olan görünüm bilgilerini dikkate alır. Ayrıca sabit hızlı hareket ve doğrusal gözlem modeli ile birlikte standart bir Kalman filtresi kullanılmaktadır. Kalman filtresi, durum uzayı ile gösterilen bir dinamik sistemde, modelin önceki bilgileriyle birlikte giriş ve çıkış bilgilerinden sistemin durumlarını tahmin edilebilen filtredir. Burada nesnelerin önceki durumlarına göre bir sonraki durumunu tahmin etmeye çalışır.



Şekil 0.5 : DeepSORT algoritması.

Gerçek çevrimiçi takip uygulamalarında öncelikle kendini çevrimdışı olarak ispatlamış iyi ayırt edici bir modele ihtiyaç vardır. Bu amaçla DeepSORT algoritmasındaki evrişimli sinir ağı, 1261 yayaya ait 1.100.000'den fazla görüntüyü içeren geniş ölçekli kimliğe sahip kişi veri seti üzerinde eğitilmiştir. Eğitilen ağın mimarisi Çizelge 3.1'de verilmiştir.

Çizelge 0.3 : DeepSORT evrişimli sinir ağı mimarisi.

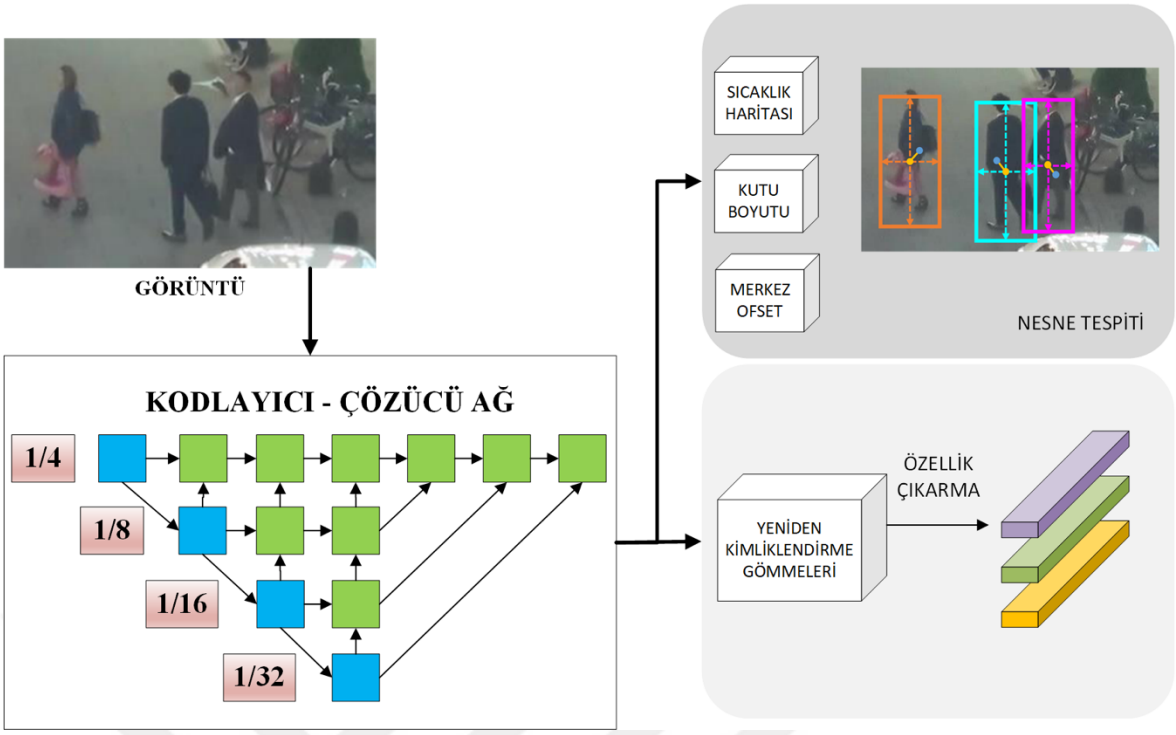
Tür	Boyut/Adım	Çıktı
Evrişim 1	3 x 3/1	32 x 128 x 64
Evrişim 2	3 x 3/1	32 x 128 x 64
Max Pool 3	3 x 3/2	32 x 64 x 32
Atık 4	3 × 3/1	32 x 64 x 32
Atık 5	3 × 3/1	32 x 64 x 32
Atık 6	3 × 3/2	64 x 32 x 16
Atık 7	3 × 3/1	64 x 32 x 16
Atık 8	3 × 3/2	128 x 16 x 8
Atık 9	3 × 3/1	128 x 16 x 8
Yoğun 10		128
Toptan and L2 Normalizasyonu		128

3.2. Çoklu Nesne Takibi Algoritmaları

Çoklu nesne takibi, görüntüler üzerinde nesnelerin takibini amaçlayan, yükseliş eğiliminde olan çalışma alanlarından biridir. Görüntü üzerinde birden fazla nesneyi aynı anda takip etmeyi hedefleyen sistemlere çoklu nesne takibi sistemleri denilmektedir. Çoklu nesne takibi genel olarak görüntü üzerinde nesnelerin tespit edildikten sonra ikinci aşamada tespit edilen nesnelerin ilişkilendirilmesi olarak kısaca ifade edile bilinir. Bu alanda çalışılan yöntemlerin başarı oranları, tespit ve ilişkilendirme aşamalarının başarısına bağlı olarak değişkenlik gösterebilmektedir. Bu iki aşamayı tek model ile çözen CenterTrack [28], FairMOT [29], ByteTrack [30] algoritmalarının yanı sıra iki aşama da çözen algoritmalar da mevcuttur. Bu iki yaklaşım arasındaki genel farklılık ikili modellerde başarı daha yüksek olurken tekli modeller gerçek zamanlı sistemlere daha uygun olmasıdır. İkili modellerin gerçek zamanlı sistemlerde daha yavaş kalmasının ana sebebi öznitelikler tespit ve ilişkilendirme aşamalarında ayrı ayrı ele alınmasıdır. Tekli model ile çözüm sağlayan yaklaşımlarda ise tespit aşamasında çıkarılan öznitelikler ilişkilendirme adımında da kullanılmaktadır.

FairMOT algoritması çoklu nesne takibi alanında tespit ve ilişkilendirme aşamaları için tek modelli bir derin öğrenme yaklaşımıdır. Model, nesne tespiti aşamasında sınır kutusu kullanmak yerine CenterNet [28] modelinde ilk defa kullanılan nesne merkezi yaklaşımını kullanmıştır. CenterNet'teki bu yeni yaklaşım, nesnenin merkez noktasını bulmayı amaçlamaktadır. Sınır kutusu kullanımının dezavantajı, nesnenin yeniden kimliklendirme aşamasında başarıyı düşürmesidir. Bir sınır kutusu çakışan birden fazla nesneye işaret edebildiği gibi birden çok sınır kutusunun tek bir nesneye işaret edebileceği ifade edilmiştir.

Ayrıca, model ile tek bir ağda işlem gerçekleştirildiği için tespit aşamasında çıkarılan öznitelikler aynı şekilde ilişkilendirme aşamasında kullanmaktadır. Özellikle, nesne tespitinde nesne sınıflarını ve konumlarını tahmin etmek için derin ve soyut özellikler gerekirken, yeniden kimliklendirme aşamasında aynı sınıfın farklı örneklerini ayırt etmek için daha çok düşük seviyeli görünüm özellikleri kullanılır. Yapılan ampirik deneylerde çok katmanlı öznitelik toplama işlemi ile bu soruna çözüm sağlanabileceği tespit edilmiştir. Yüksek boyutlu ilişkilendirme özniteliklerinin çıkartılması nesnelere aynı kimliği verme işleminde başarıyı yükseltse de diğer taraftan nesne tespitinin doğruluğunu düşürdüğü belirtilmiştir. Bu durum ile baş edebilmek için düşük boyutlu öznitelikler kullanılmıştır. Ağda düşük boyutlu özniteliklerin kullanılması ayrıca aşırı öğrenme riskini azaltmaktadır ve öğrenme hızını arttırmaktadır. Ayrıca, omurga ağı olarak DLA-34 [32] kullanılmıştır. Modele, DLA-34 omurga ağı sayesinde çok katmanlı öznitelik toplama özelliği kazandırılmıştır. Sırasıyla ısı haritalarını, nesne merkezi ofsetlerini ve sınırlayıcı kutu boyutlarını tahmin etmek için DLA-34'e üç paralel kafa eklenir. DLA-34 modelinde yeniden kimliklendirme adımı, nesneleri ayırt edebilecek özellikler üretmeyi amaçlar. Aynı nesneler arasındaki yakınlık farklı nesneler arasındaki yakınlıktan daha büyük olmalıdır. Bu amaca ulaşmak için, her konuma ait yeniden tanımlama özellikleri çıkarılarak omurga özelliklerinin üstüne 128 çekirdekli bir evrişim katmanı uygulanmaktadır. Modelin mimari çizimi Şekil 3.2'de verilmiştir. Şekil 3.2 kodlayıcı-çözücü ağ olarak gösterilen DLA-34 omurga ağının tasarımını ifade etmektedir. Şekil 3.2'deki diğer çizimler nesne tespiti ve yeniden kimliklendirme aşamalarına ait akışları göstermektedir.



Şekil 0.6 : FairMOT mimarisi.

Çoklu nesne takibinde birçok yöntem, puanları bir eşikten daha yüksek olan algılama kutularını ilişkilendirerek kimlikleri elde eder ve algılama puanı düşük olan tıkanmış nesneler saf dışı kalır. Tıkanmış nesnelerin saf dışı kalması, gerçek nesne kayıplarını ve parçalanmış yörüngelerin oluşmasına sebep olur. Bu sorunu çözmek için, yalnızca yüksek puanlı olanlar yerine her algılama kutusu takip edilerek basit, etkili ve genel bir ilişkilendirme yöntemi ByteTrack [30] modelinde sunulmuştur. Düşük puana sahip tespit kutuları için, gerçek nesneleri bulmak ve arka plan algılamalarını filtrelemek amacıyla izlerdeki benzerlikler kullanılmaktadır. Tek bir V100 GPU üzerinde 30 FPS çalışma hızı ile MOT17 [33] test setinde 80.3 MOTA, 77.3 IDF1 ve 63.1 HOTA değerleri elde edilmiştir. ByteTrack modelinde Byte adında yeni ilişkilendirme algoritması önerilmiştir. Bu algoritmada önceden yüksek skor almış nesnelerin sonraki görüntülerde eşik değerin altında kalması bu nesnelerin kimlik kaybına uğramasına sebep olmaktadır. Bundan dolayı öncelikle yüksek puanlı nesne tespit kutuları hareket benzerliğine göre mevcut takipler ile eşleştirilir. Hareket benzerliği, tahmin edilen kutu ile nesne tespit kutusunun kesişimi vasıtasıyla hesaplanır. Sonra yeni görüntüdeki takiplerin yerini tahmin etmek için Kalman filtresi kullanılır. Çoklu nesne takibinin gelişmiş performansını öne çıkarmak için, yüksek performanslı YOLOX [34] nesne tespit detektörünü ilişkilendirme yöntemi BYTE ile donatarak ByteTrack çoklu nesne takip modeli sunulmuştur. Çizelge 3.2'de MOT17

doğrulama veri setinde SORT [31], DeepSORT [9], MOTDT [35] ve yeni önerilen BYTE ilişkilendirme algoritmalarının karşılaştırması bulunmaktadır. Çizelge 3.2'de belirtildiği gibi BYTE algoritmasının diğer algoritmalara göre daha başarılı sonuçlar verdiği görülmektedir [30].

Çizelge 0.4 : MOT17 doğrulama veri setinde ilişkilendirme algoritmalarının karşılaştırması [30]

Algoritma	Yeniden Kimliklendirme	MOTA	IDF1	IDs	FPS
SORT		74.6	76.9	291	**30.1**
DeepSORT	X	75.4	77.2	239	13.5
MOTDT	X	75.8	77.6	273	11.1
BYTE		**76.6**	**79.3**	**159**	29.6

3.3. Çoklu Nesne Takibinde Performans Metrikleri

2008 yılında Clear MOT [36] performans metrikleri çoklu nesne takibi algoritmalarının başarı durumlarını analiz edebilmek için önerilmiştir. Bu metrikler oluşturulurken temelde iki kriter dikkate alınmıştır. Geliştirilecek çoklu nesne takibi yaklaşımının, nesne konumlarını belirlemede izleyicinin hassasiyetini değerlendirmeye izin vermesi gerektiği vurgulanmıştır. Ayrıca nesne konfigürasyonlarını zaman içinde tutarlı bir şekilde izleyen, yani nesne yörüngelerini doğru bir şekilde takip edebilen, her nesne için tam olarak bir yörünge üretme yeteneğini göstermesi gerektiği ifade edilmiştir. Bu kriterlere ek olarak;

- Sonuçların karşılaştırılabilir ve değerlendirmeler basitleştirilebilir olması için mümkün olduğunca az sayıda serbest parametreye ve ayarlanabilir eşiklere sahip olmasının,
- Açık, kolay anlaşılır ve özellikle farklı türlerde birden fazla hatanın meydana gelmesi veya sekans boyunca hataların eşit olmayan şekilde yeniden bölünmesi durumunda insan sezgisine göre davranmasının,
- Çoğu tip izleyicinin (2B, 3B izleyiciler, nesne merkezi izleyicileri veya nesne alanı izleyicileri) karşılaştırılmasına izin verecek kadar genel olmasının,
- Sayıca az ve yine de anlamlı olmaları için, örneğin birçok sistemin karşılaştırıldığı büyük değerlendirmelerde kullanılabilir olmasının

önemli olabileceği belirtilmiştir. Yukarıda sıralanan kriterlere uygun olacak şekilde iki adet denklem sunulmuştur. Bunlardan ilki çoklu nesne takibinde hassasiyet (Multi Object Tracking Precision (MOTP)):

$$MOTP = \frac{\sum_{i,t} h_t^i}{\sum_t b_t} \tag{0.1}$$

Denklem 3.1 mevcut çerçeve için b_t, t anındaki bulunan eşleşme sayısını, h_t^i, t anındaki nesne-hipotez toplam uzaklığını ifade etmektedir.

İkinci olarak tanımlanan denklem çoklu nesne takibinde doğruluk (Multi Object Tracking Accuracy (MOTA)):

$$MOTA = \frac{\sum_t (n_t + yp_t + nne_t)}{\sum_t g_t} \tag{0.2}$$

Denklem 3.2 mevcut çerçeve için n_t, t anındaki eksik tespit edilen sayısını, yp_t, t anındaki yanlış pozitiflerin sayısını, nne_t, t anındaki eşleşmeyenlerin sayısı ifade etmektedir [36].

Ergys ve arkadaşları [37] çoklu hedef ve çoklu kamera takibi için 2016 yılında yeni performans metrikleri sunmuşlardır. Çok hedefli, çok kameralı izleme sistemlerinde ilerlemeyi hızlandırmak amacıyla her türlü hatayı tek tip olarak ele alan ve hata kaynakları yerine doğru tanımlamayı vurgulayan yeni bir çift hassasiyet duyarlılık performans ölçümü sunmuşlardır. Çoklu kamera ölçümlerinin kimlik eşleşme performansını doğru bir şekilde hesaba kattığı belirtilmiştir. Denklem 3.3'te kimlik hassasiyeti (Identification precision (IDP)), denklem 3.4'te kimlik duyarlılığı (Identification Recall (IDR)), denklem 3.5'te ise kimlik F1 skoru (Identification F1 Score (IDF1)) olarak tanımlanmıştır.

$$IDP = \frac{IDTP}{IDTP + IDFP} \tag{0.3}$$

$$IDR = \frac{IDTP}{IDTP + IDFN} \tag{0.4}$$

$$IDF1 = \frac{2IDTP}{2IDTP + IDFP + IDFN} \tag{0.5}$$

(Identification True Positive (IDTP)) kimlik doğru pozitifleri, (Identification False Positive (IDFP)) kimlik yanlış pozitifleri, (Identification False Negative (IDFN)) kimlik yanlış negatifleri ifade etmektedir. Gerçek kimliklerin hesaplanan kimlikler ile bire bir örtüşmesi gerçek sonuç eşleşmesi olarak tanımlanır. Gerçek sonuç eşleşmesine dayalı performans değerlendirme yaklaşımı ile daha önce bahsedilen tüm zayıflıklar basit ve tek tip

bir şekilde ele alınmaktadır. Ayrıca her türlü hata aynı ölçü birimiyle, yani yanlış atanan veya atanmayan görüntü sayısıyla ölçümlendirilir.

Diğer bir çalışmada, çoklu nesne takibinde takipçileri karşılaştırmak için doğru tespit, ilişkilendirme ve lokalizasyon etkisini birleşik bir metrikte açıkça dengeleyen değerlendirme metriği, yüksek sıralı takip doğruluğu (Higher Order Tracking Accuracy(HOTA)) [38] algoritması önerilmiştir. HOTA, beş temel hata türünün her birini ayrı ayrı değerlendirebilen ve takip performansının net analizini sağlayan alt metriklerden oluşan bölümlere ayrışır. HOTA'nın MOT performans değerlendirme metriklerine ek olarak daha önce dikkate alınmamış önemli yönleri sunulmuştur. Ayrıca, HOTA izleme performansının, insan sezgisel değerlendirmesiyle daha uyumlu olduğu ifade edilmiştir. HOTA, MOTA ve IDF1'deki her iki hatayı da açıkça ölçer ve bunları dengeli bir şekilde birleştirir. HOTA algoritması, MOTA ve IDF1'de bulunmayan izleme sonuçlarının yerelleştirme doğruluğunu da ölçmektedir.

Çizelge 0.5 : HOTA sezgisel anlam çizelgesi [38]**.**

gt		DetA	MOTA	HOTA	IDF1	AssA
A	100	%50	%50	%50	%67	%50
B	50	%70	%69	%50	%52	%35
C	35 35	%100	%97	%50	%25	%25
	25 25 25 25					
		Artan Tespit Ölçümü			Artan İlişkilendirme Ölçümü	

Çizelge 3.3 HOTA'nın sezgisel olarak anlaşılması için verilmiştir. MOTA, çoklu nesne takip doğruluğunu, IDF1 ise kimlik F1 skorunu göstermektedir. Tespit doğruluğu DetA, sadece hizalama tespitlerinin yüzdesidir. İlişkilendirme doğruluğu AssA, tüm tespitler üzerinden ortalaması alınan, eşleşen yörüngeler arasındaki ortalama hizalamaları ifade etmektedir. Nihai HOTA puanı, farklı yerelleştirme eşikleri üzerinden ortalaması alınan bu iki puanın geometrik ortalamasıdır.

$$HOTA_{\alpha} = \sqrt{\frac{\sum_{c \varepsilon \{TP\}} A(c)}{|TP| + |FN| + |FP|}} \tag{0.6}$$

$$A(c) = \frac{|TPA(c)|}{|TPA(c)| + |FNA(c)| + |FPA(c)|} \tag{0.7}$$

Denklem 3.6 lokalizasyon eşikleri üzerinden α yerelleştirme eşik değeri için $HOTA_{\propto}$ entegrasyon denklemini temsil etmektedir. Denklem 3.7'deki $A(c)$ ise ilişkilendirme skorunu ifade etmektedir. $A(c)$ hesaplanırken ilişkilendirme başarısını değerlendirmek için, doğru pozitif ilişkilendirme (True Positive Association (TPA)), yanlış negatif ilişkilendirme (False Negative Association (FNA)) ve yanlış pozitif ilişkilendirme (False Positive Association (FPA)) gibi yeni kavramlar önerilmiştir. $HOTA_{\alpha}$ denklemi hem tespit hem de ilişkilendirme doğruluğunu hesaba katar, ancak yerelleştirme doğruluğunu hesaba katmaz. HOTA'nın lokalizasyonu ölçmesi için denklem 3.8'de verilen nihai HOTA puanı, 0 ile 1 değer aralığında geçerli α değerleri için hesaplanan $HOTA_{\alpha}$ puanının integralidir.

$$HOTA = \int_0^1 HOTA_{\alpha} d_{\alpha} \tag{0.8}$$

Denklem 3.9'da gösterilen yaklaşık HOTA değeri ise 0.05 ile 0.95 arasındaki α değerleri için 0.05 aralıklar ile toplanan $HOTA_{\alpha}$ sonuçlarının aritmetik ortalaması alınarak elde edilir.

$$HOTA \cong \frac{1}{19} \sum_{n=0.05,0.1,\ldots,0.95} HOTA_{\alpha} \tag{0.9}$$

3.4. Optimizasyon Algoritmaları

Çok farklı optimizasyon algoritmaları vardır. Bunlar ile ilgili bilgiye ilgili kaynaklardan ulaşılabilir. Bu bölümde yapılan çalışmada kullanılan optimizasyon algoritmaları açıklanacaktır.

Stokastik gradyan inişi (SGİ) gradyan inişin (Gİ) bir türüdür. Gradyan inişi ile aynı performansı üretir. Gradyan inişi ile aynı performansı üretmesinin sebebi, öğrenme oranın düşük olmasıdır. Makine öğrenim algoritmalarında stokastik gradyan inişi, uygun parametre değerlerini keşfetmek veya uygun düzgünlük özelliklerine sahip bir amaç fonksiyonunu iyileştirmek için yinelemeli bir algoritmadır. Parametre değerlerinde sık sık küçük ayarlamalar yaparak hata tahminini azaltmayı amaçlar. Bu yöntem, parçalı eğitim veri grupları üzerinde gradyanı hesaplayan ve parametre değerlerini optimize eden gradyan inişinin stokastik bir versiyonu olarak sunulmaktadır. Büyük veri kümeleri için gradyan inişi pek uygun değildir ve yavaş çalışır. Bunun sebebi, her yinelemede tüm eğitim verileri için yeniden hesaplama yapılmasıdır. Stokastik gradyan inişinin en büyük avantajı, kolay uygulanabilir ve verimli olmasıdır [27].

Gradyan inişinin dezavantajlarıyla baş edebilmek için Rprop, hata fonksiyonunun davranışına göre ağırlık güncellemelerinin yerel bir uyarlamasını gerçekleştirir. Rprop yöntemini diğer yöntemlere göre daha başarılı yapan, uyarlama sürecinde türev boyutunun bulanıklaşmamasıdır. Sonuç olarak, Rprop şeffaf ve verimli bir uyum sürecine yol açar [39].

Geoffrey Hinton tarafından keşfedilen RMSprop, parçalı öğrenme için Rprop [39], algoritmasının yeni bir versiyonu olarak düşünülebilir. RMSprop salınımları, momentumdan farklı bir şekilde sönümlemeye çalışmaktadır. Ayrıca, RMSprop öğrenme oranının ayarlanma ihtiyacını ortadan kaldırmaktadır. Her parametre için RMSprop'da farklı bir öğrenme oranı seçilir [40].

Adam algoritması, stokastik amaç fonksiyonlarının birinci dereceden gradyan temelli optimizasyonu için uyarlanabilir ve düşük dereceli tahminlere dayalı olarak sunulmuştur. Hesaplama açısından çok etkili olan bu algoritmanın uygulanması da oldukça basittir. Çok az bellek gereksinimine ihtiyaç duyması algoritmayı daha avantajlı bir konuma yükseltmektedir. Büyük veri veya fazla parametreler bakımından çok uygun bir yöntemdir. Ek olarak yöntem, çok gürültülü sabit olmayan hedefler veya seyrek gradyanlı problemler için de uygundur. Algoritmanın sahip olduğu hiper parametreler tipik olarak çok az ayarlama gerektirir. Deneysel sonuçlar, Adam'ın pratikte iyi çalıştığını ve diğer stokastik optimizasyon yöntemlerine kıyasla daha olumlu bir yöntem olduğunu göstermiştir [41].

4. YÜZ TESPİTİ

Günlük hayatta kullandığımız bilgisayarlar ve diğer elektronik cihazlar, bir kişinin yüz ifadelerini yeterince iyi yorumladıklarında daha kullanıcı dostu hale gelerek insan - makine sistemlerini geliştirmeye fırsat verir.

Serbest ortamda yüz tespiti ve hizalama, çeşitli pozlar, aydınlatmalar ve oklüzyonlar nedeniyle zordur. Derin öğrenme yaklaşımları yüz tespiti ve hizalamadaki zorluklar ile baş etmede etkileyici performans elde etmektedirler. Yüz tespit ve hizalamada en çok kullanılan MTCNN [10], yüz tespiti ve hizalanması arasındaki doğal korelasyondan yararlanan, derin ve kademeli çoklu görevi çatısında barındıran, başarılı bir performans gösteren derin öğrenme modelidir. MTCNN, yüz dönüm noktası konumunu hasas bir şekilde tahmin etmek için dikkatlice tasarlanmış derin evrişimsel ağların üç aşamasına sahip kademeli bir mimariden yararlanır. Model, yüz tespitinde WIDER FACE [42] ve yüz hizalama için LFW [43] veri setleri üzerinde gerçek zamanlı olarak üstün doğruluk sağlamaktadır.

Çoklu görev gerçekleştirebilen basamaklı evrişimsel ağlardan oluşan MTCNN modeli, yüz tespiti ve hizalama konusunda etkileyici sonuçlar göstermiştir. Yüz tespiti ve hizalama arasındaki korelasyonu miras alan MTCNN temelde üç bölümden oluşur [10]:

(1) İlk bölümde, aday pencerelerini oluşturmak için (Proposal Network (P-Net)) ağından yararlanır. Bu ağ sonrasında iki amaç için kullanılır. Birinci amaç, aday pencereler üzerinde yüz olan ve yüz olmayanları sınıflandırmaktır. İkinci olarak, yüz konumuna sınırlayıcı kutu regresyon vektörlerini ve maksimum olmayan bastırma (Non-maximum suppression(NMS)) aday birleştirmesini tahmin etmek için kullanılır.

(2) İkinci bölümde, (Refine Network (R-Net)), P-Net'ten farklı olarak bölge tekliflerini çıkarmak yerine birçok yanlış adayı elemektedir.

(3) Son bölümde ise (Output Network (O-Net)), yapısal olarak R-Net'e benzemek ile birlikte yüzün beş adet karekteristik noktalarının konumu çıkarmaktadır.

Diğer bir çalışmada RetinaFace [11], yüz kutusu tahminini, 2B yüz karekteristik nokta lokalizasyonunu ve 3B köşe regresyonunu ortak bir çatı altında birleştirirek yeni bir model sunmuştur. Model eğitilirken verileri arttırmak için kullanıma açık veri setleri üzerinde bir takım işlemler yapılmıştır. İlk olarak WIDER FACE [42] veri setinde yüzün beş adet karekteristik noktası manuel olarak işlenmiştir. Sonrasında WIDER FACE, AFLW [44] ve FDDB [45] veri setlerindeki yüz görüntüleri için 3B tepe noktaları oluşturmak üzere yarı

otomatik bir işleme hattı kullanılmıştır. Gerçekleştirilen veri arttırım ek işlemlerinde, 3B yüz rekonstrüksiyonu için ortak 3B topoloji sınırlandırılmış görüntü düzleminde 3B tepe noktalarını öngören bir regresyon hedefi sunulmaktadır. Deneysel sonuçlarda, RetinaFace'in aynı anda kararlı yüz tespitini, doğru 2B yüz hizalamasını ve sağlam 3B yüz rekonstrüksiyonunu tek aşamalı model ile verimli şekilde başardığı gözlemlenmiştir.

Yüz tespitinde büyük adımlar atılmış olsa da, düşük hesaplama maliyeti ve yüksek hassasiyet ile verimli bir yüz tespitine tam olarak ulaşılamamıştır. SCRFD [12] bu sorunlara çözüm olabilmesi için başarılı bir modeldir. Model için iki basit ama etkili yöntem sunulmuştur. Bunlar aşağıda sırasıyla verilmiştir.

(1) Denek veri setlerinin istatistiklerine bağlı olarak en çok ihtiyaç duyulan aşamalar için eğitim veri setindeki görüntü sayılarını artıran örneklem yeniden dağıtım yaklaşımı kullanılmıştır.

(2) Özenle tanımlanmış bir arama metodolojisine dayalı olarak, modelin farklı bileşenleri (omurga, boynu ve başı) arasındaki hesaplamayı yeniden tahsis eden hesaplama yeniden dağıtım yöntemi kullanılmıştır.

WIDER FACE üzerinde yürütülen detaylı deneyler, çok çeşitli hesaplama formlarında önerilen SCRFD ailesi için verimlilik-doğruluk dengesine çok önem verilmiştir. Önerilen alt model SCRFD-34GF, TinaFace [46] modelini %3,86 oranında geride bırakırken, VGA çözünürlüklü görüntülerde 3 kattan daha hızlı bir performans sunmaktadır.

Yüz tespiti veri seti WIDER FACE üzerindeki ortalama hassasiyet (OH), TinaFace modelinde %92.4'e ulaşılmıştır. TinaFace kısıtlamasız yüz tespitinde etkileyici sonuçlara ulaşsa da büyük maliyet oluşturan hesaplamalar yapmaktadır. SCRFD modelinde, hesaplama maliyetini azaltmak amacı ile test için büyük ölçek kullanmak yerine sabit bir VGA çözünürlüğü (640×480) altında verimli yüz tespiti yapılmıştır. Bu ölçek ayarı altında, WIDER FACE'deki yüzlerin çoğu 32×32 pikselden küçüktür ve bu nedenle sığ aşamalarla tahmin gerçekleşir. Tahmini daha iyileştirmek için daha fazla eğitim örneği elde etmenin faydalı olacağı düşünülmüştür. Eğitim veri setindeki görüntü sayısını arttırmak için büyük bir görüntü kırpma stratejisi ile örnek yeniden dağıtım yöntemi kullanılmıştır.

Ayrıca, bir yüz dedektörünün yapısı, hesaplama dağılımını oluşturmaktadır ve doğruluğu ile verimliliğini belirlemede anahtar niteliğindedir. Modelde hesaplama dağılımının ilkeleri tekrardan incelenmiştir ve serbestlik dereceleri kontrol altına alınarak arama alanı azaltılmıştır. Modelin farklı bileşenlerinden omurga, boyun ve kafa üzerinde farklı

konfigürasyonlara sahip mimariler rastgele örneklem alındıktan sonra modellerin istatistiklerine dayanarak, önyüklemeyi hesaplayan ve en iyi modellerin düştüğü olası aralığı tahmin eden bir çalışma yapılmıştır. Sonuç olarak bir yüz dedektörünün farklı bileşenleri (omurga, boyun ve kafa) arasında hesaplama yeniden dağıtım yöntemi ile basitleştirilmiş bir arama alanı tasarlanmıştır.

5. YÜZ TANIMA

Yüz tanıma işlemi için yüzün gerçek özniteliklerinin doğru şekilde çıkarılması gerekmektedir. Bunun için literatürde yer alan başarılı yöntemler arasında SphereFace [13], CosFace [14], ArcFace [15] gibi derin öğrenme modelleri gösterile bilinir.

SphereFace [13], belirli bir metrik uzayında ideal yüz özniteliklerinin minimum sınıflar arası mesafeden daha küçük maksimum sınıf içi mesafeye sahip olmasının beklendiği, açık küme derin yüz tanıma problemini ele almaktadır. Açık küme yaklaşımında eğitim ile test veri setindeki kimlikler birbirinden tamamen farklı olması beklenmektedir. Bu yaklaşım ile yüz tanıma sisteminde yüzün öznitelikleri çıkarıldıktan sonra en yakın komşu metriği kullanılarak karşılaştırma yapılmaktadır. Kapalı kümede ise test veri seti içerisinde eğitim veri setine ait kimlikler bulunmak zorundadır. Sonraki aşamada tahmin işlemi kapalı model üzerinden gerçekleştirilir. Bununla birlikte, açık küme sisteminde öğrenme özelliklerinin elde edilmesi, büyük sınıf içi varyasyon ve yüksek sınıflar arası benzerlik sebebiyle genellikle zordur. Bu nedenle, evrişimli sinir ağlarının açısal olarak ayırt edici özellikleri öğrenmesini sağlayan açısal Softmax olarak nitelendirilen A-Softmax kayıp fonksiyonu kullanılmıştır. A-Softmax kayıp fonksiyonu, sınıflandırma işleminde softmax kaybına benzer şekilde birden çok sınıf için kolayca genelleştirilebilinir. A-Softmax kayıp fonksiyonu, karar bölgelerini optimize ederek yani daha ayrı hale getirerek aynı anda sınıflar arası marjı genişletir ve sınıf içi açısal dağılımı sıkılaştırır. A-Softmax kaybı net bir geometrik yoruma sahiptir. A-Softmax kaybı tarafından öğrenilen özellikler, bir hiper küre zemini üzerindeki jeodezik mesafeye eşdeğer olan ayırt edici açısal mesafe metriğini oluşturur. Ayrıca, A-Softmax kaybı, bir m parametresi aracılığıyla açısal marjı nicel ayarlamalar ile nicel analiz yapmamıza olanak sağlar. Sınıflar arası minimum mesafenin sınıf içi maksimum mesafeden daha büyük olması görevini öğrenebilmesi amacıyla *m* için alt sınırlar elde edilmiştir. Yapılan çalışma neticesinde ikili sınıflandırma için minimum *m* değeri $2 + \sqrt{3}$ den büyük ve çoklu sınıflandırma için minimum *m* değeri 3 den büyük veya eşit olması gerektiği ispatlanmıştır. Çalışmanın devamında daha büyük *m* değerinin sürekli olarak daha iyi çalıştığını ve *m*=4 değerinin genellikle yeterli olacağı ifade edilmiştir. Denklem 5.1'de A-Softmax kayıp fonksiyonu verilmiştir.

$$L_{\text{ang}} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{\|x_i\|\varphi(\theta_{y_i,i})}}{e^{\|x_i\|\varphi(\theta_{y_i,i})} + \sum_{j\neq y_i} e^{\|x_i\|\cos(\theta_{j,i})}} \tag{0.10}$$

Denklemde $\varphi(\theta_{y_i,i}) = (-1)^k \cos(m\,\theta_{y_i,i}) - 2k,\ \theta_{y_i,i} \in \left[\frac{k\pi}{m}, \frac{(k+1)\pi}{m}\right]$ ve açısal kenar boşluğunun boyutunu kontrol eden bir tamsayı $k \in [0, m-1]. m \geq 1$ olarak formüle edilmiştir. x_i, i'ninci örneğin y_i'ninci sınıfa ait derin özniteliklerini temsil etmektedir. m değeri marjini ve θ_i açısı W_i ile x_i arasındaki açıyı ifade etmektedir.

Yapılan iyileştirmelerden sonra CASIA-WebFace veri seti [47] üzerinde eğitilen SphereFace, LFW [43], YTF [48] ve MegaFace [49] veri setlerinde çeşitli kriterlerde rekabetçi sonuçlar elde etmiştir. A-Softmax kullanılarak eğitilen yüz tanıma modeli SphereFace, LFW'de %99.42 , YTF'de %95 ve MegaFace Rank-1'de ise %75.76 ile yüz tanıma doğrulukları elde etmiştir.

Yüz doğrulama ve tanımlama dahil olmak üzere yüz tanımanın temel görevi, yüz özelliklerini ayırt etmeyi içerir. Bununla birlikte, derin evrişimli sinir ağlarından daha iyi sonuç alabilmek için geleneksel Softmax yerine merkez kaybı, büyük marjlı softmax kaybı ve açısal softmax kaybı gibi kayıp fonksiyonları literatürde yerini almıştır. Tüm bu geliştirilmiş kayıp fonksiyonları, sınıflar arası varyansı en üst düzeye çıkarmak ve sınıf içi varyansı en aza indirmek amacıyla sunulmuşlardır. CosFace [14] modelinde ise, bu fikir farklı bir perspektiften bakılarak yeni bir kayıp fonksiyonu olan büyük marjlı kosinüs kaybı (Large Margin Cosine Loss (LMCL)) önerilmiştir. Daha spesifik olarak, açısal uzayda karar marjını daha da maksimize etmek için bir kosinüs marjı teriminin dahil edildiği radyal varyasyonları ortadan kaldırmak için hem özellikleri hem de ağırlık vektörlerini L2 formunda normalleştirerek yeniden formüle edilmiştir. Sonuç olarak, normalizasyon ve kosinüs karar marjı maksimizasyonu sayesinde minimum sınıf içi varyans ve maksimum sınıflar arası varyans elde edilmiştir. Büyük marjlı kosinüs kayıp fonksiyonu kullanılarak CASIA-WebFace veri seti [47] üzerinde eğitilmiş CosFace modeli, MegaFace [49], YTF [48] ve LFW [43] gibi en popüler halka açık yüz tanıma veri setleri üzerinde kapsamlı değerlendirmeler yapılmıştır.

$$L_s = -\frac{1}{N}\sum_{i=1}^{N} log\, p_i = -\frac{1}{N}\sum_{i=1}^{N} log \frac{e^{f_{y_i}}}{\sum_{j=1}^{C} e^{f_j}} \tag{0.11}$$

Denklem 5.2'de p_i, x_i'nin doğru şekilde sınıflandırılmasının arkasındaki olasılığı belirtir. N eğitim örneklerinin sayısıdır. C sınıfların sayısıdır. f_j, W_j ağırlık vektörü ve B_j yanlılık vektörü ile tam bağlantılı bir katmanın aktivasyon fonksiyonudur. Hesaplamalarda kolaylık olması için $B_j = 0$ kabul edilierek denklem 5.3'te f_j şu şekilde verilir.

$$f_j = W_j^T x = \|W_j\| \|x\| \cos\theta_j \tag{0.12}$$

Denklem 5.3'teki θ_j açısı W_j ile x arasındaki açıyı ifade etmektedir. Bu denklem ile vektörlerin hem normunun hem de açısının arkasındaki olasılığa katkı sağlanmaktadır.

Etkili öğrenmeyi geliştirmek için, W'nin normu mutlaka değişmez olmalıdır. Bunun için L2 normalizasyonunu uygulanarak $\|W_j\| = 1$ elde edilmiştir. Test aşamasında, test edilen bir yüz çiftinin yüz tanıma puanı genellikle iki özellik vektörü arasındaki kosinüs benzerliğine göre hesaplanır. Bu durum, öznitelik vektörü x'in, puanlama işlevine katkıda bulunmadığı tespit edildiğinden, eğitim aşamasında $\|x\| = \sigma$ sabitlenmiştir. Sonuç olarak, sınıflandırılmasının arkasındaki olasılık sadece açının kosinüsüne dayanır. Değiştirilmiş kayıp fonksiyonu denklem 5.4'te formüle edilmiştir.

$$L_{\text{ns}} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\,cos(\theta_{y_i,i})}}{\sum_{j=1}^{C} e^{s\,cos(\theta_{y_i,i})}} \tag{0.13}$$

Denklem 5.4'deki kayıp fonksiyonu softmax kaybının normalleştirilmiş versiyonu olarak sunulmuştur. Ancak, normalleştirilmiş versiyon tarafından öğrenilen özellikler yeterince ayırt edici değildir çünkü normalleştirilmiş versiyon sadece doğru sınıflandırmayı vurgulamaktadır. Ayırt edici özelliğini geliştirmek için, softmax'ın kosinüs formülasyonuna kosinüs marjı m sınıflandırma sınırı olarak dahil edilir. Sonuç olarak denklem 5.5'te büyük marjlı kosinüs kayıp fonksiyonu son haline ulaşır.

$$L_{\text{lmc}} = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\,(cos(\theta_{y_i,i}) - m)}}{e^{s\,(cos(\theta_{y_i,i}) - m)} + \sum_{j \neq y_i}^{n} e^{s\,cos(\theta_{j,i})}} \tag{0.14}$$

Büyük marjlı kosinüs kayıp fonksiyonu kullanılarak CASIA-WebFace veri seti [47] üzerinde eğitilmiş CosFace modeli, MegaFace [49], YTF [48] ve LFW [43] gibi en popüler halka açık yüz tanıma veri setleri üzerinde kapsamlı değerlendirmeler yapılmıştır. CosFace, LFW'de %99.73 , YTF'de %97.6, MegaFace Challenge 1 Rank-1'de ise %84.26 ve MegaFace Challenge 2 Rank-1'de %77.06 ile yüz tanıma doğruluklarını elde etmiştir.

Yüz tanıma için derin evrişimli sinir ağlarında öğrenmedeki ana zorluklardan biri ayırt edici gücü artırabilen uygun kayıp fonksiyonlarının tasarımı olduğu şüphesiz birçok çalışmada vurgulanmaktadır. ArcFace modeli de aynı yaklaşım ile hareket ederek yeni bir kayıp fonksiyonu kullanmıştır. Yüksek düzeyde ayırt edici öznitelikler elde etmek için önerilen yüz tanıma modeli ArcFace [15], yeni kayıp fonksiyonu ile bir hiperküre üzerindeki jeodezik mesafeye tam olarak uyması nedeniyle net bir geometrik yoruma sahiptir. Denklem 5.6'da verilen standart birçok yüz tanıma yaklaşımında kullanılan softmax kayıp fonksiyonu tekrar yorumlanmıştır. N toptan sayısını, n sınıf numarasını ifade etmektedir. x_i, i'ninci

örneğin y_i'ninci sınıfa ait derin özniteliklerini temsil etmektedir. W_j değeri, W ağırlığın j'ninci sütununa işaret eder. b_j ise yanlılık değerini temsil etmektedir.

$$L_1 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{W_{y_i}^T x_i + b_{y_i}}}{\sum_{j=1}^{n} e^{W_{y_i}^T x_i + b_j}} \tag{0.15}$$

Yeni iyileştirmeler eklenerek oluşturulan denklem 5.7'de önerilen kayıp fonksiyonunda yanlılık değeri sıfır olarak kabul edilir. Ayrıca, $W_j^T x_i = \|W_j\| \|x_i\| \cos\theta_j$ dönüşümündeki W_j ile x_i arasında kalan θ açısı kullanılarak $cos\theta$ değeri kayıp fonksiyonuna dahil edilmiştir. Dönüşümde W_j değeri, $l2\ formu\ ile$ normalleştirilerek $\|W_j\| = 1$ elde edilir ve ayrıca $\|x_i\|$ gömme özellik değeri aynı şekilde $l2$ formu ile normalleştirildikten sonra s değeri ile ölçeklendirilir. Normalleştirme adımı, tahminlerin yalnızca öznitelik ile ağırlık arasındaki açıya bağlı olmasını sağlar ve öğrenilen gömme özellikler böylece s yarıçaplı bir hiperküre üzerinde dağıtılır.

$$L_2 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\,cos\theta_{y_i}}}{e^{s\,cos\theta_{y_i}} + \sum_{j=1, j\neq y_i}^{n} e^{s\,cos\theta_j}} \tag{0.16}$$

ArcFace modelinde ortaya çıkan gömme özellikler hiperküredeki her bir özellik merkezi etrafında dağılmaktadır. Her biri kendi özellik merkezi etrafında dağılan gömme özellikler için sınıf içi yoğunluğu ve sınıflar arası uyuşmazlığı aynı anda geliştirmek amacıyla denklem 5.8'de x_i ve gerçek referans değer ağırlığı W_{y_i}arasına bir ek açısal m marj değeri eklenir.

$$L_3 = -\frac{1}{N}\sum_{i=1}^{N} log\frac{e^{s\,cos(\theta_{y_i}+m)}}{e^{s\cos\left(\theta_{y_i}+m\right)} + \sum_{j=1, j\neq y_i}^{n} e^{s\,cos\theta_j}} \tag{0.17}$$

ArcFace, birçok yüz çiftini içeren büyük ölçekli görüntü veri tabanını ve büyük ölçekli bir video veri kümesini yorumlayan yüz tanıma modelleri ile kıyaslanmıştır. Diğer gelişmiş yüz tanıma yöntemlerine karşı kapsamlı deneysel değerlendirmelerin sonucunda ArcFace'in sürekli olarak iyi performans gösterdiği ve ihmal edilebilir hesaplama yükü ile kolayca uygulanabileceği ifade edilmiştir. MS-Celeb-1M veri setinin yarı otomatik rafine versiyonu olan MS1MV2 veri setinde eğitimi gerçekleştirilen ArcFace, LFW [43] veri setinde %99.83, YTF [48] veri setinde %98.02 doğruluğa ulaşmıştır.

6. APACHE KAFKA

Apache Kafka [16], birçok şirket tarafından yüksek performanslı veri boru hatları, akış analitiği, veri entegrasyonu ve kritik görev uygulamaları için kullanılan açık kaynaklı bir dağıtık olay akış platformudur. Apache Kafka, videolar, veri tabanları, sensörler, mobil cihazlar, bulut hizmetleri ve yazılım uygulamaları gibi olay kaynaklarından gerçek zamanlı olarak veri yakalama uygulamasıdır. Bu olay akışlarını daha sonra geri almak için dayanıklı bir şekilde depolanmaktadır. Ayrıca, olay akışlarını gerçek zamanlı ve geçmişe dönük olarak işleme ve olay akışlarını gerektiği gibi farklı hedef teknolojilere yönlendirme en önemli görevleri arasından yer almaktadır. Böylece olay akışı, doğru bilginin doğru yerde, doğru zamanda olması için verilerin sürekli akışını ve yorumlanmasını sağlar. Kafka, olay akışı için kullanım senaryoları uçtan uca test edilmiş tek bir çözümle uygulanabilmesi için aşağıda belirtilen üç temel özelliği birleştirir:

- Verilerin diğer sistemlerden sürekli aktarılması olayını içeren olay akışlarını yayınlamak (yazmak) ve bunlara abone olmak (okumak) en önemli özelliğidir.
- Olay akışlarını istenilen kadar dayanıklı ve güvenilir bir şekilde depolanması diğer bir özelliğidir.
- Olay akışları oluştukça veya geriye dönük işlenmesine izin vermesi başka bir özelliğidir.

Tüm bu işlevsellik, dağıtılmış, yüksek düzeyde ölçeklenebilir, esnek, hataya dayanıklı ve güvenli bir şekilde sağlanır. Kafka, yalın donanım, sanal makineler ve kapsayıcılar ile şirket içinde veya bulutta devreye alınabilir.

Kafka, yüksek performanslı bir TCP ağ protokolü aracılığıyla iletişim kuran sunucular ve istemcilerden oluşan dağıtık bir sistemdir. Kafka sunucuları, birden çok veri merkezine veya bulut bölgesine yayılabilen bir veya daha fazla sunucu kümesi olarak çalıştırıla bilinir. Bu sunuculardan bazıları, aracılar olarak adlandırılan depolama katmanını oluşturur. Geriye kalan diğer sunucular ise görev açısından kritik kullanım senaryolarını uygulamanıza izin vermek için bir Kafka kümesi ile yüksek düzeyde ölçeklenebilir ve hataya dayanıklıdır. Sunucularından herhangi biri arızalanırsa, kalan sunucular, bir veri kaybı olmadan akışı devam ettirmek için sorumluluğu üstlenirler. Kafka, olay akışlarını paralel, ölçeklenebilir ve ağ sorunları veya makine arızaları durumunda bile hataya dayanıklı bir şekilde okuyan, yazan ve işleyen dağıtılmış uygulamalar ve mikro servisler yazmanıza olanak tanır. Kafka

istemcilerinin, Java, Scala, Go, Python, C/C++ ve diğer birçok programlama dilleri için desteği mevcuttur.

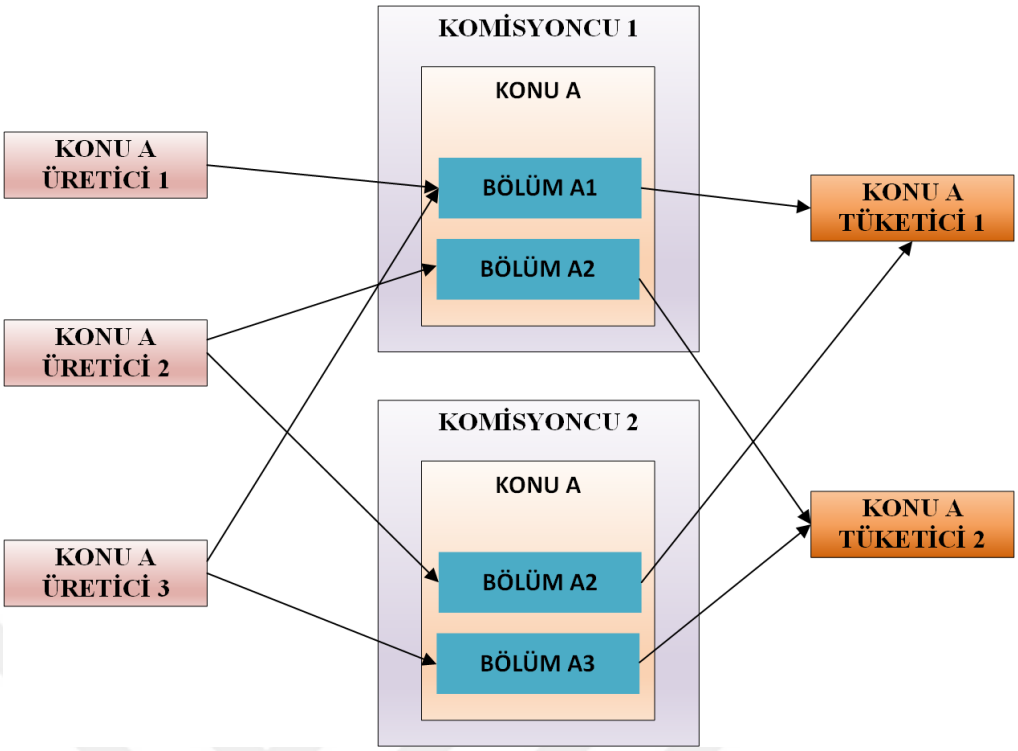
Kafka'da veri okuma veya yazma işlemi olay kavramı şeklinde yapılmaktadır. Kavramsal olarak, bir olayın anahtarı, değeri, zaman damgası ve isteğe bağlı meta veri başlıkları vardır. Örneğin:

- Olay Anahtar: "Kamera-1"
- Olay Değeri: "0x012031EF323…"
- Olay Zaman Damgası: "2021-12-09T01:23:23"

Kafka da üreticiler, olayları Kafka'ya yazan istemci uygulamalarıdır. Tüketiciler ise, bu olaylara abone olarak okuyan ve işleyen uygulamalardır. Kafka'da üreticiler ve tüketiciler birbirlerinden ayrı ve bağımsız olarak çalışırlar. Bu durum, Kafka'nın bilinen yüksek ölçeklenebilirliğini elde etmesi için önemli bir tasarım kalıbıdır. Mesela, üreticilerin hiçbir zaman tüketicileri beklemesine gerek yoktur. Kafka, olayların sadece bir kez işlenmesi gibi çeşitli garantiler sağlamaktadır.

Olaylar, konularda kalıcı bir şekilde saklanacak biçimde organize edilir. Yani, bir konu bir dosya sistemindeki bir klasöre ve olaylar klasördeki dosyalara benzetilebilir. Geleneksel kuyruk mesajlaşma sistemlerinin aksine, olaylar tüketimden sonra belirli bir süre silinmez. Kafka'nın olayları konu başına yapılandırma ayarı aracılığıyla ne kadar süreyle tutulması gerektiği tanımlanır ve tanıma göre eski olaylar silinir. Kafka'nın performansı, veri boyutuna göre etkin bir şekilde sabittir.

Bir konu farklı Kafka komisyonculara yayılabilmektedir. Komisyoncular aslında verilerinizin bu dağıtılmış yerleşimini ölçeklenebilir yapmak için çok önemlidir, çünkü istemci uygulamalarının aynı anda birçok aracıdan veri okumasına ve yazmasına olanak tanır. Bir konuda yeni bir olay yayınlandığında, konunun bölümlerinden birine eklenir. Kafka, aynı olay anahtarına sahip olayları aynı bölüme yazar. Kafka, bölümdeki konunun her zaman yazıldığı sırayla okuyacağını tüketiciye garanti eder. Şekil 6.1'de aynı konu için farklı komisyonculara ve farklı bölümlerde üretici ve tüketicinin örnek veri akışı çizilmiştir.



Şekil 0.7 : Apache Kafka üretici, tüketici arasından örnek veri akışı.

7. SOCKET.IO

Socket.IO, sunucu ile tarayıcı arasında gerçek zamanlı, çift yönlü ve olay tabanlı iletişimi sağlayan bir kütüphanedir. Java, C++, Swift, Python gibi birçok programa dili desteğine sahiptir. Socket.IO kütüphanesi, WebSocket API'sinin etrafındaki hafif bir sarmalayıcı olarak oluşturulmuştur. Ayrıca Socket.IO, düz bir WebSocket nesnesine ek olarak aşağıda listelenen ek özellikleri sağlar [17].

- Güvenilirlik ile WebSocket bağlantısının kurulamaması durumunda HTTP uzun yoklama seçeneğinin kullanılabilmesine olanak sağlar.
- Otomatik yeniden bağlantı kurulmasını destekler.
- Paketleri ara belleğe alır.
- Paketin ulaşıp ulaşmadığı bilgisine sahiptir.
- Bütün istemcilere yayım yapabilir.
- Uygulama mantığını tek bir paylaşılan bağlantı üzerinde bölebilir.

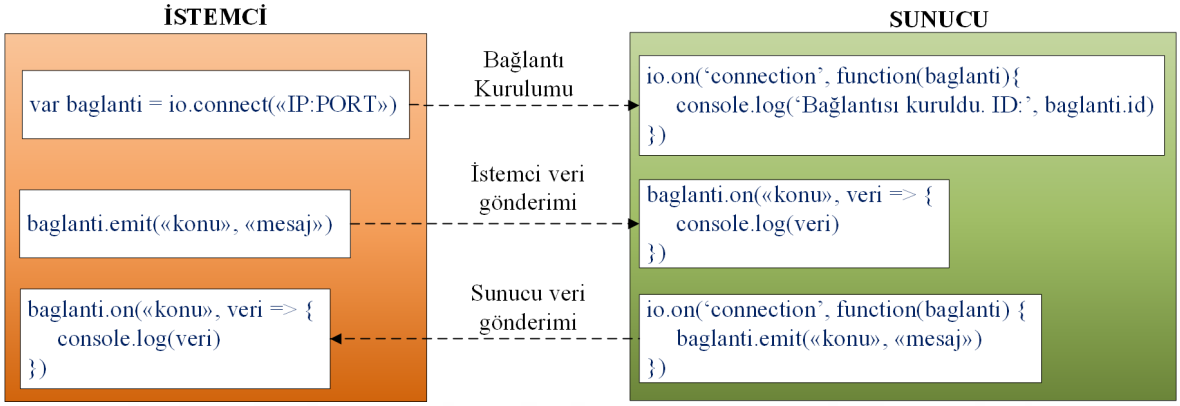
Socket.IO'nun düşük seviyeli implementasyonu Engine.IO ile Socket.IO'nun içindeki üst düzey API yani Socket.IO'nun kendisi olmak üzere kod tabanı iki farklı katmana ayrılmıştır.

Socket.IO istemcisi, HTTP uzun yoklama aktarımıyla bağlantısını varsayılan olarak kurar. Bunun asıl sebebi, iki yönlü bir iletişim kurmanın en iyi yolu WebSocket olsa da, kurumsal vekil sunucular, kişisel güvenlik duvarı, antivirüs yazılımları nedeniyle WebSocket bağlantısı kurmak her zaman mümkün olmayabilir. Kullanıcı bakımından, başarısız bir WebSocket bağlantısı, gerçek zamanlı uygulamanın veri alışverişine başlaması için en az 10 saniye bekleme süresi demektedir. Bu durum algısal gerçek zamanlı uygulamalarda kullanıcı deneyimine zarar verir. Bundan dolayı uzun yoklama adımı hemen hemen her yerde çalışır, bu nedenle ilk başta kullanılır, böylece hemen bir bağlantı elde edilir. Ardından arka planda, uzun yoklama bağlantısını bir WebSocket bağlantısına yükseltme girişiminde bulunulur. Yükseltme başarılı olursa, uzun yoklama durur ve oturum WebSocket bağlantısına geçer. Başarılı olmazsa, uzun yoklama bağlantısı açık kalır ve kullanılmaya devam eder. Aşağıda belirtilen sırayla bağlantı oluşturulur.

- İlk olarak el sıkışma işlemi yapılır (Sonraki isteklerde kullanılacak oturum kimlik bilgisi).

- HTTP uzun yoklama veri gönderme işlemi yapılır.
- HTTP uzun yoklama ile veri alma işlemi yapılır.
- WebSocket bağlantısına yükseltme işlemi yapılır.
- Eğer WebSocket bağlantısı başarılı şekilde kurulduysa HTTP uzun yoklama kapatılır ve WebSocket ile devam edilir aksi halde HTTP uzun yoklama ile devam edilir.

Şekil 7.1'de Socket.IO veri trafiğinin kodlama şeması verilmiştir. Şekilde dne gösterildiği gibi bağlantı için io.connect, veri göndermek içi baglanti.emit, veri almak için ise sunucu baglanti.on metotları kullanılır.



Şekil 0.8 : Socket.IO veri trafiği.

Socket.IO'da bir HTTP isteği başarısız olduğunda, WebSocket bağlantısı kapandığında veya baglanti.disconnect metodu çağrıldığı durumlarda bağlantının kapalı olduğu kabul edilir.

Socket.IO aktarım katmanı olarak WebSocket'i kullanmasına rağmen, her pakete ek meta veriler ekler. Bu nedenle, bir WebSocket istemcisi bir Socket.IO sunucusuna başarılı bir şekilde bağlanamayacaktır ve aynı şekilde bir Socket.IO istemcisi de düz bir WebSocket sunucusuna bağlanamayacaktır.

8. UYGULAMALAR

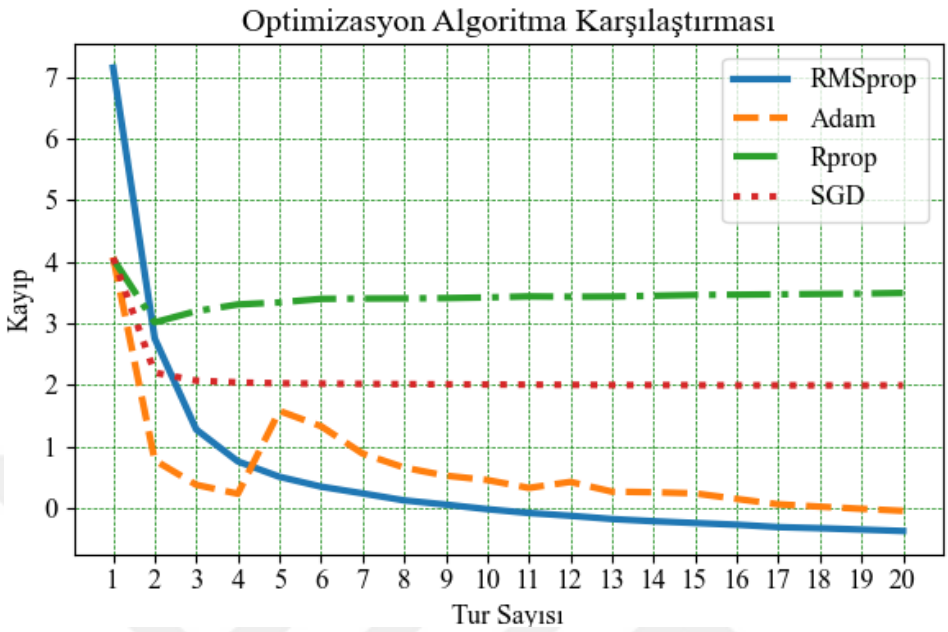
Bu bölümde iki ayrı çalışma yapılmıştır. Gerçekleştirilen birinci çalışmada hiper parametre optimizasyon algoritmalarının çoklu nesne takip modeli FairMOT [29] üzerindeki etkileri incelenmiştir. İkinci çalışmada ise çoklu kamera kullanılan ortamlarda derin öğrenme modellerinin yardımıyla gerçek zamanlı yüz takibinin yapılması hedeflenmiştir. Mevcut çalışmada kameralardan alınan görüntüler dağıtık olay akış sistemi Apache Kafka [16] üzerinden kayıt altına alındıktan sonra işleyici zincirinden geçirilerek çıkarılan veriler veri tabanına kaydedilmiştir. Ayrıca son kullanıcının web sitesi üzerinden talep etmesi durumunda Socket.IO kullanılarak işlenen görüntünün sunulması amaçlanmaktadır. 8.1. bölümünde ilk çalışmaya ait detaylara yer verilirken 8.2. bölümünde ise ikinci çalışma ile ilgili uygulama detayları paylaşılmıştır.

8.1. Hiperparametre Optimizasyon Algoritmalarının Karşılaştırılması

FairMOT [29] derin öğrenme modelinin, gerçek zamanlı çoklu nesne takibi alanında başarılı bir çalışma olduğu gözlemlenmiştir. Model üzerinde yapılan çalışma ile MOT20 [50] doğrulama veri setindeki doğruluk başarısının arttırılması hedeflenmiştir. MOT20 eğim veri seti yaya görüntülerinden oluşmaktadır. Veri seti farklı ortam, çözünürlük ve yaya yoğunluk seviyelerine göre MOT20-01, MOT20-02, MOT20-03, MOT20-05 olmak üzere dört gruba ayrılmıştır. Çalışmada MOT20 veri seti %70 eğitim için %30 ise doğrulama seti olarak bölünmüştür.

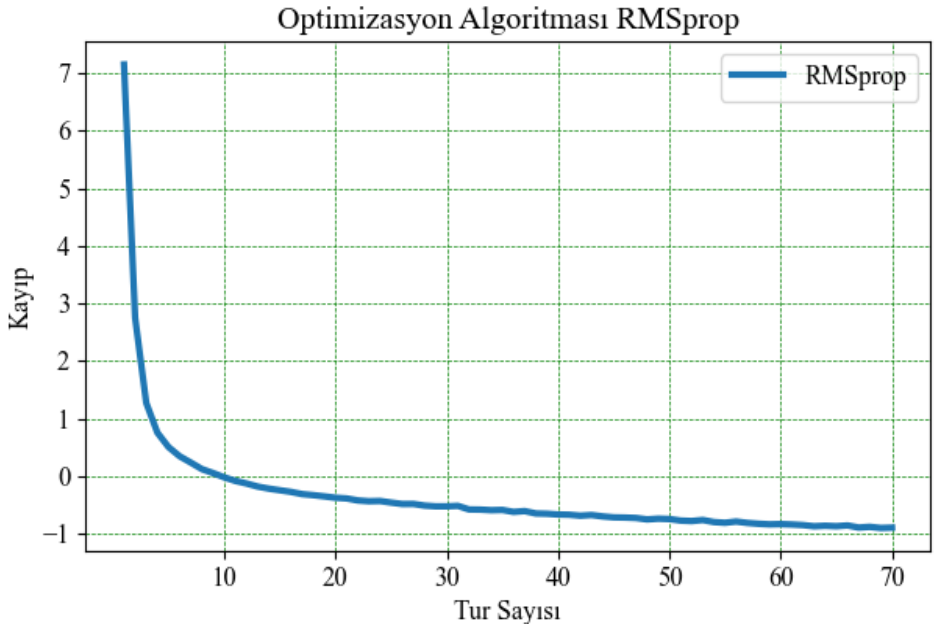
FairMOT modelinin ince ayar eğitim aşamasında hiper parametre optimizasyon algoritmaları üzerinde çalışma yapılmıştır. RMSprop[40], Adam [41], Rprop[39] ve SGİ [27] optimizasyon algoritmaları kullanılarak gerçekleştirilen çalışmanın amacı FairMOT modelinde en düşük kayıp değerine sahip optimizasyon algoritmasını bulmaktır. Eğitim, MOT20 eğitim veri seti üzerinde gerçekleştirilmiştir. Her bir optimizasyon algoritması 20

tur eğitilerek ilk aşama tamamlanmıştır. Şekil 8.1'de ifade edildiği gibi RMSprop algoritmasının en düşük kayıp değerine sahip olduğu tespit edilmiştir.



Şekil 0.9 : Optimizasyon algoritmalarının karşılaştırması.

MOT20 eğitim veri seti üzerinde RMSprop algoritması kullanılarak 50 tur daha eğitilen modelin kayıp değerinin daha da düştüğü Şekil 8.2'de gösterilmiştir.



Şekil 0.10 : RMSprop algoritması ile 70 tur eğitim.

RMSprop algoritması kullanılarak 70 tur eğitilen modelin, MOT20 doğrulama veri seti üzerinde %76.7 başarı elde ettiği tespit edilmiştir. Çizelge 8.1'de yapılan çalışmanın MOT20 doğrulama veri seti üzerindeki performans değerlendirme verilerine yer verilmiştir [4]. Çizelge 8.1'deki *MOTA, MOTP, IDF1, IDP, IDR* sütunlarına ait denklemler bölüm 3.3'te verilmiştir. Çizelge 8.1'de her bir sütun için hesaplanan *TOPLAM* satırı ise veri seti gruplarının ortalamasını ifade etmektedir. *TOPLAM* satırı hesaplanırken her bir grupta bulunan videolardaki görüntü çerçeve sayısına göre ortalama alınmıştır.

Çizelge 0.6 : MOT20 doğrulama veri seti değerlendirmesi.

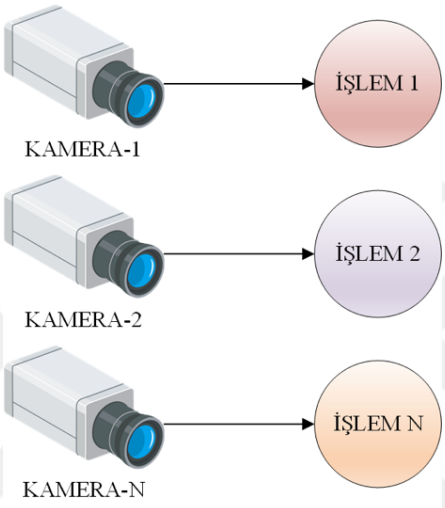
Veri Seti	MOTA	MOTP	IDF1	IDP	IDR
MOT20-01	67.9%	0.160	76.4%	91.1%	65.8%
MOT20-02	69.6%	0.153	69.4%	81.8%	60.2%
MOT20-03	77.9%	0.173	85.7%	91.5%	80.7%
MOT20-05	78.4%	0.165	84.7%	90.8%	79.3%
TOPLAM	76.7%	0.166	82.6%	89.8%	76.5%

8.2. Derin Öğrenme Tabanlı Gerçek Zamanlı Kimliklendirme Sistemi

Gerçekleştirilen çalışmada derin öğrenme teknikleri kullanılarak çoklu kamerada gerçek zamanlı yüz takibi sistem uygulaması yapılmıştır. Her kameradan akan görüntünün birbirinden bağımsız bir şekilde ve eş zamanlı işlenmesi sağlanmıştır. Aynı zamanda ölçeklenebilir ve hata toleranslı bir sistem olarak kurgulanmıştır. Ayrıca, yüz takibi adımı, kararlı, verimli ve az maliyetli olarak bir arada sunulmuştur. Yüz takibi adımında DeepSORT takip algoritmasına yüz tanıma modeli eklenerek yeni bir yaklaşım elde edilmiştir. Yüz tanıma modelinde görüntüdeki yüzler ile veri tabanındaki kişi yüz görüntüleri karşılaştırılarak görüntüdeki yüze ait kişi tespiti yapılmıştır. Tespit edilen kişi ve tespit edildiği kamera ile birlikte veri tabanında depolanmıştır. Böylece, gerçekleşen yüz tanıma işlemi sonucunda kişinin tespit edilmesine kimliklendirme denir. Baştan sona gerçek zamanlı olarak tasarlanan sistem üzerinde çift yönlü bir iletişim hattı kurularak işlenmiş görüntüler son kullanıcı ile paylaşılmıştır.

Sistemde ilk olarak herbir kamera için işletim sistemi seviyesinde ayrı bir işlem başlatılır. Ayrı ayrı işlemlerin başlatılması ve yönetilmesi için yönetim kod dosyası çalıştırılarak yapılır. Yönetim kod dosyası Linux sunucuda servis olarak çalıştırılır. Yönetim kod dosyası, Apache Kafka konularını, üreticilerini, tüketicilerini kurulu zaman aralıklarında tekrar kontrol edip yönetmektedir. Yönetim kod dosyası, konunun oluşturulması, üretici

veya tüketici işlemi durmuş ise tekrar çalıştırılması ile görevlidir. Konu, her bir kamera için PostgreSQL [51] veri tabanında yer alan birincil anahtar değerleri ile oluşturulmaktadır. Şekil 8.3'te gösterildiği gibi işletim sistemi seviyesinde başlatılan herbir işlem sadece bir kamera verisinin işlenmesinden sorumlu tutularak eş zamanlı çalışması sağlanmıştır. Kamera kayıtları ve kameralara ait ayarlar PostgreSQL veri tabanında tutulmuştur. Son kullanıcı tarafından aktif olarak işaretlenen kameralar veri tabanında sorgulandıktan sonra herbiri için eş zamanlı işlem başlatılır.



Şekil 0.11 : Kamera verilerinin eş zamanlı işlenmesi.

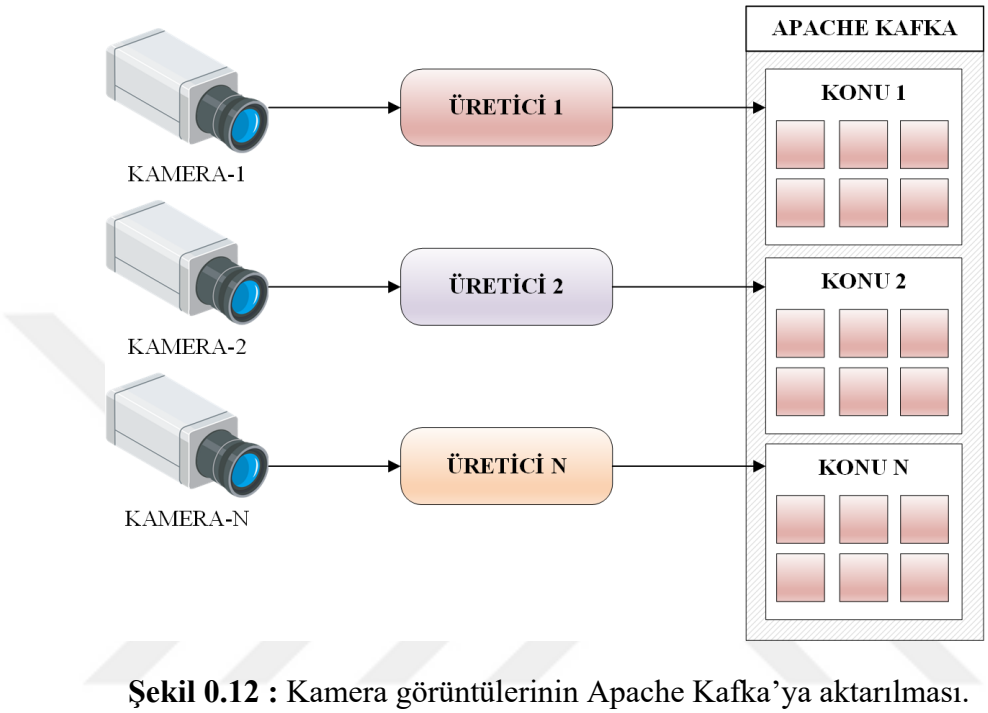
Gerçek zamanlı akan görüntülerin işlenmeden önce veri kaybını engellemek için tampon bir kayıt alanı gerekmekte olduğu farkedilmiştir. Bunun için gerçek zamanlı akan görüntü verisini ön bellekleme yapabilmek için olay akış işleme sistemine ihtiyaç duyulmuştur. Ayrıca görüntülerin ölçeklenebilir şekilde ele alınması gerekliliği düşünülerek, olay akış işleme sistemi olarak Apache Kafka [16] kullanılmıştır. Apache Kafa seçilmesindeki ana sebepler, gerçek zamanlı veri akışlarının işlenmesi için birleşik, yüksek verimli, düşük gecikme süreli ve ölçeklenebilir bir platform sunmasıdır.

Uygulama dört aşamada gerçekleştirilmiştir.

- Kameradan alınan görüntünün akış işleme sistemine kaydedilmesi ilk aşamadır.
- Akış işleme sisteminden alınan görüntünün oluşturulan sorumluluk zincir yapısına verilmesi ikinci aşamadır.
- Yüzü tanımlanan kişinin veri tabanına kaydedilmesi üçüncü aşamadır.

- Web sitesinde kameranın işlenmiş görüntüsünün gösterilmesi ise son aşamadır.

Çalışmanın ilk aşaması Şekil 8.4'te belirtildiği gibi görüntülerin akışı esnasında her kamera için ayrı üretici işlemi çalıştırılır. Üretici aldığı görüntüyü kameranın veri tabanındaki birincil anahtar değerine sahip konuya gönderir.



Şekil 0.12 : Kamera görüntülerinin Apache Kafka'ya aktarılması.

Çizelge 8.2'de belirtilen üretici kodu ile kameralardan alınan ham video görüntüler Apache Kafka konusuna gönderilir. Üretici kodda ilk olarak Apache Kafka üretici nesnesi oluşturulur. Sonrasında OpenCV [52] kütüphanesindeki *read* fonksiyonu yardımıyla kameradan görüntü alınır. Okunan görüntü OpenCV *imencode* fonksiyonu kullanılarak *jpeg* görüntü formatına dönüştürülür. Son olarak *jpeg* formatına dönüştürülen görüntünün bayt dizisi Apache Kafka konusuna gönderilir.

Çizelge 0.7 : Üretici kod implementasyonu.

```
import cv2
from confluent_kafka import Producer
import kafka.consts

producer = Producer({'bootstrap.servers':
",".join(kafka.consts.KAFKA_BOOTSTRAP_SERVERS)})

cam = cv2.VideoCapture(URL)
```

```
while True:
    ret_val, img = cam.read()
    if ret_val:
        encodedImg = []
        res, encodedImg = cv2.imencode('.jpg', img)

        if res:
            producer.poll(0)
            producer.produce(topic, encodedImg.tobytes())
```

İkinci aşamada Apache Kafkada bulunan görüntüler, her bir kamera için oluşturulan tüketiciler vasıtasıyla işlenmek için okunur. Çizelge 8.3'te verilen tüketici kodu Apache Kafka sunucusuna bağlantı kurduktan sonra görüntünün alınacağı konuya abone olur ve işlenmesi için sonraki adımlara görüntüyü gönderir. Kafka tüketicisi *'auto.offset.reset': 'earliest'* ayarı ile ofseti ilk gelen değerden başlayarak okunması için hazırlanır. Ayrıca Kafka tüketici grubu olarak *cameragroup* ayarlanır. Tüketici kod, konuya gelecek olan yeni görüntüleri sürekli dinlemektedir. Görüntünün *jpeg* baytları Kafka konusundan alındıktan sonra OpenCV *imencode* fonksiyonu ile tekrar ilk formatına dönüştürülür ve sorumluluk zincirinin ilk işleyicisine gönderilir.

Çizelge 0.8 : Tüketici kod implementasyonu.

```
import cv2
import numpy as np
from confluent_kafka import Consumer
import kafka.consts

consumer = Consumer({
      'bootstrap.servers': ",".join(kafka.consts.KAFKA_BOOTSTRAP_SERVERS),
      'group.id': 'cameragroup',
      'auto.offset.reset': 'earliest' })
consumer.subscribe([topic])

while True:
    msg = consumer.poll(0)
    if msg is None:
        continue
    if msg.error():
        print("Consumer error: {}".format(msg.error()))
        continue
```

```
    nparr = np.frombuffer(msg.value(), np.uint8)
    frame = cv2.imdecode(nparr, cv2.IMREAD_COLOR)
    baslangicIsleyici.calistir(frame)
```

Tüketici tarafından Apache Kafka konusundan alınan görüntüler sorumluluk zincirinde sırayla işlenmektedir. Bu aşamada sorumluluk zincir örüntüsünün seçilmesindeki asıl amaç, yeni çıkarımlar ekleme esnekliği sağlamaktır. Böylelikle sisteme istendiği zaman yaş tespiti, cinsiyet tespiti vb. modeller kolaylıkla entegre edilebilir. Sorumluluk zincirindeki işleyici nesneleri, ata sınıf olarak Çizelge 8.4'te yer verilen *Isleyici* sınıfı kullanılarak oluşturulur. *Isleyici* sınıfındaki *sonrakini_ayarla* fonksiyonu ile sonraki işleyici zincire eklenir. *Isleyici* sınıfındaki *calistir* fonksiyonu, görüntü verisi ve opsiyonel parametreler ile çalışmaktadır. Bu fonksiyon, işleyici üzerinde yapılmak istenen iş mantığını içermektedir.

Çizelge 0.9 : Sorumluk zincir örüntü ata sınıf kod implementasyonu.

```
from __future__ import annotations
from abc import ABC, abstractmethod
from typing import Any

class AbstractIsleyici(ABC):

    @abstractmethod
    def sonrakini_ayarla(self, isleyici: Isleyici) -> Isleyici:
        pass

    @abstractmethod
    def calistir(self, goruntu: Any):
        pass

class Isleyici(AbstractIsleyici):

    _sonraki_isleyici: Isleyici = None
    def sonrakini_ayarla(self, isleyici: Isleyici) -> Isleyici:
        self._sonraki_isleyici = isleyici
        return isleyici

    @abstractmethod
    def calistir(self, goruntu: Any, **kwargs):
        if self._sonraki_isleyici:
            return self._sonraki_isleyici.calistir(goruntu, **kwargs)
        return None
```

Mevcut sistemde sorumluluk zinciri dört adet işleyiciden oluşmaktadır. Bir sonraki işleyiciye veriyi gönderecek biçimde Çizelge 8.5'te belirtilen kod ile sorumluluk zinciri oluşturulur. *isleyicileri_ayarla* fonksiyonu, *işleyiciler* listesini parametre olarak alır. *isleyiciler* listesinde verilen sıraya göre sorumluluk zinciri oluşturulur.

Çizelge 0.10 : İşleyicileri ayarlama implementasyonu.

```
def isleyicileri_ayarla(isleyiciler):
    onceki_isleyici = None
    baslangic_isleyici = None
    for isleyici in isleyiciler:
        if onceki_isleyici is not None:
            onceki_isleyici.sonrakini_ayarla(isleyici)
        else:
            baslangic_isleyici = isleyici
        onceki_isleyici = isleyici
    return baslangic_isleyici

isleyicileri_ayarla([YuzTespitIsleyici, YuzTakipIsleyici, WebSoketIsleyici,
VeritabaniIsleyici])
```

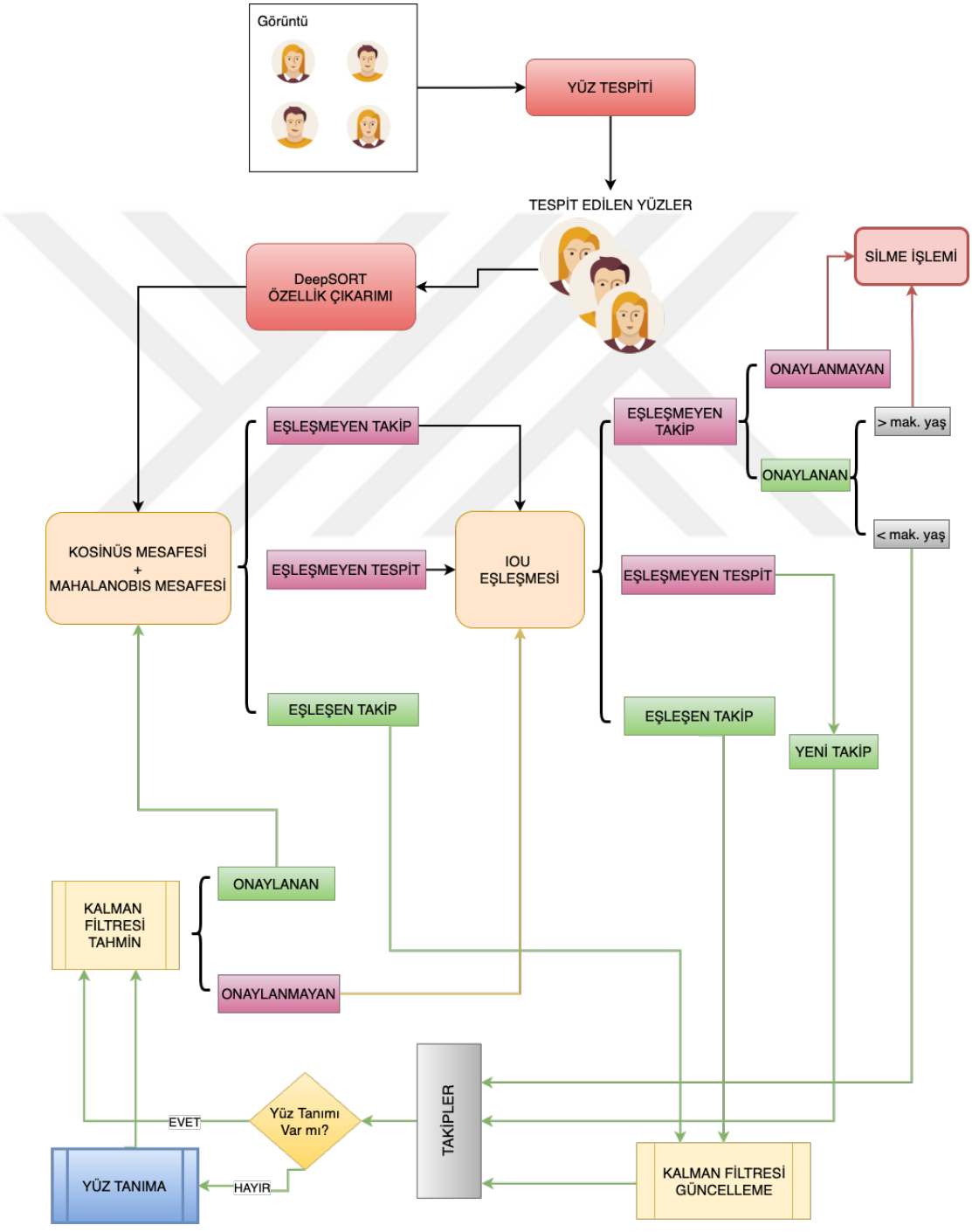
Şekil 8.5'te belirtildiği gibi sırayla zincir işlemler kullanılmaktadır. Yüz tespiti için Yüz Tespit İşleyici ve yüz takibi için Yüz Takip İşleyici zincirin giriş işlemleri olarak kullanılır. Web sitesine görüntünün gönderimi amacıyla paylaşımlı belleğe görüntünün kaydedilmesi için Web Soket İşleyici kullanılır. Veri tabanına kişi logunun kaydedilmesi amacıyla "database" konusuna öz bilgi gönderimi için Veri Tabanı İşleyici son zincir işlem olarak kullanılır.

Şekil 0.13 : İşleyici zinciri.

İlk işleyici olan Yüz Tespit İşleyicisi, verimli ve yüksek doğruluk değerine sahip SCRFD [12] modelini kullanmıştır. Yüz tespiti için görüntü modele verildikten sonra çıktı olarak yüz sınır kutuları elde edilir. Model girişine verilecek görüntüler 640x640 olarak

tekrardan boyutlandırılır. Elde edilen yüz sınır kutuları bir sonraki adım olan Yüz Takip İşleyici zincir işlemini beslemektedir.

Yüz Takip İşleyicisinde, DeepSORT [9] çoklu nesne takip algoritması, yüz tanıma derin öğrenme modeli ArcFace [53] ile birleştirilerek Şekil 8.6'da gösterilen yeni bir yaklaşım sunulmuştur.



Şekil 0.14 : Yüz takip modeli.

Yeni tasarımda ilk olarak yüz sınır kutuları kullanılarak görüntüdeki yüzlerin DeepSORT [9] modelinde yer alan evrişimli sinir ağı vasıtasıyla özellikleri çıkarılır. 2.8 milyon parametreye sahip evrişimli sinir ağı iki evrişim katmanı ve altı artık bloktan oluşmaktadır. 128 çıkış özellik haritası, yoğun katmanda hesaplandıktan sonra toptan işlem ve *l2* form normalleştirmesi ile oluşturulur. Çıkarılan özellikler en yakın komşu mesafe ölçümü ile daha önce tespit edilen yüzler ile takip edilen yüzlerin eşleştirilme durumu sorgulanır. En yakın komşu mesafe ölçümü için Kosinüs ve Mahalanobis hesaplamaları birleştirilerek Macar algoritması ile atama işlemi yapılarak gerçekleştirilir. Sonrasında eşleşmeyen tespit veya takipler daha önce doğrulanan takipler ile görüntüdeki kesişimleri hesaplanarak tekrar eşleşme testine tabi tutulurlar. Bu adımdan sonra eşleşmeyen takiplerden onaylanmayan ve maksimum yaşını dolduranlar silinir. Diğer durumdakiler ise takibe alınmaya başlanır veya takipleri devam ettirilir.

DeepSORT yaklaşımında bir görüntüdeki nesneye kimlik verilebilmesi için en az 3 defa görünmesi gerekmektedir. Nesne, 3 defa görünene kadar "*onaylanmayan takip"* durumunda kalır. Nesne 3 defa göründükten sonra "*onaylanan takip*" olarak işaretlenir aksi halde "*eşleşmeyen takip"* durumuna düşer ve takipten çıkarılır. "*onaylanan takip*" durumundaki bir takibin takipten çıkarılması bir yaş değerine bağlanmıştır. "*onaylanan takip*" durumundaki bir takip, maksimum yaş değeri kadar görüntülerde tespit edilemediyse nesne takipten çıkarılır. Maksimum yaş 70 olarak belirlenmiştir. Eşleştirmeler için maksimum uzaklık 0.2, NMS maksimum çakışma değeri 0.3, maksimum birleşimlerin kesişim değeri 0.7 olarak belirlenmiştir.

Kullanılan yeni tasarım DeepSORT derin öğrenme modeline yüz sınır kutuları girdi olarak verildikten sonra ilk defa tespit edilen yüzler veya yüz tanıma işlemi daha önce başarısız olanlar için yüz tanıma modeli kullanılarak kimlikleri tespit edilmeye çalışılır. Yüz tanıma modeli olarak ArcFace kullanılır. ArcFace, kendini kanıtlamış ve yüksek doğruluk değerine sahip olduğu için tercih edilmiştir. Her bir yüz, 112x112 giriş boyutu ile ağa verilmektedir. Sonrasında yüz için gömme özellikler 512 boyutunda elde edilmektedir. ArcFace modelinde omurga ağı olarak ResNet50 veya ResNet100 kullanılmıştır. ResNet50 ağı yaklaşık 25 milyon parametreye sahipken, ResNet100 ağı yaklaşık 44 milyon parametreye sahiptir. Diğer taraftan DeepSORT evrişimli sinir ağı 2.8 milyon ile daha az parametreye sahiptir. Bundan dolayı maliyeti düşürüp verimliliği arttırmak amacıyla yüz, bir kere başarılı olarak tanımlandıktan sonra diğer araçlar vasıtasıyla takip işlemi gerçek

zamanlı olarak gerçekleştirilir. DeepSORT girişinde daha az parametreye sahip bir modelin kullanılması gerçek zamanlı takip içinde büyük avantaj sağlamaktadır.

Yüz tanıma işlemi daha önce veri tabanına kaydedilen kişilere ait yüz gömme özellikleri ile karşılaştırılarak yapılmaktadır. Bu işlemin hızlı olması için kişiye ait bilgiler bellekte sözlük anahtar değer yapısında tutulur. Yüz tanıma işlemi, yeni takip aşamasında başarısız olursa her 10 görüntüde bir toplam 5 kere denenecek şekilde tekrarlanır. Kurgulanan takip sistemi sayesinde her görüntü için yüz tanıma algoritmasının çalıştırılmasına gerek kalmamakla birlikte daha kararlı bir takip işlemi gerçekleştirilmesine olanak sağlamaktadır.

Yüz Takip İşleyicisinde işlenmiş görüntü, Web Soket İşleyicisine aktarılmaktadır. Web Soket İşleyici ile görüntü base64 formatına dönüştürüldükten sonra anahtar değer ikilisiyle paylaşımlı bellekte saklanır. Anahtar olarak kameranın birincil anahtar değeri kullanılırken, anahtara karşılık gelecek değer verisi olarak görüntüden elde edilen base64 verisi atanır. Paylaşımlı bellek alanı her kameranın güncel görüntüsüyle doldurulur. Web sitesinde kamera kontrol ekranı açıldığında kullanıcının görüntüleme yetkisinin olduğu kameraların herbiri için Socket.IO [17] kütüphanesi ile çift taraflı bir iletişim kanalı kurularak görüntü akışı sağlanır.

Web Soket İşleyicisi vasıtasıyla bellekte paylaşılan işlenmiş görüntüler Socket.IO sunucu uygulaması ile web sitesine gönderilir. Çizelge 8.5'te verilen sunucu kod gerçekleştirimi kullanılarak web sitesinden adres yolu bilgisi ile Socket.IO sunucusuna bağlantı kurulduktan sonra ilgili kamera için gönderim yapılmaya başlanır. Socket.IO Python programlama dili implementasyonu olan *flask_socketio* kütüphanesi kullanılmıştır. İlk olarak *SocketIO* sınıfı ile nesne oluşturulur. Sonra *run* fonksiyonu *adres* ve *port* bilgisi ile çalıştırılarak bağlanacak istemcileri beklemeye başlar. Adres yol bilgisi ile bağlantı kuran istemcilerin her biri eş zamanlı olarak *goruntuyu_webe_gonder* fonksiyonunda karşılanır. Son olarak, *emit* fonksiyonu kullanılarak paylaşımlı bellekteki sözlükten ilgili istemciye veri gönderimi yapılır. İstemci, iletişim kanalında adres yol bilgisi olarak kameranın birincil anahtar değerini kullanmaktadır.

Çizelge 0.11 : Socket.IO sunucu kod implementasyonu.

```
import collections
```

```
from flask import Flask, request
from flask_socketio import SocketIO, join_room, leave_room, emit

app = Flask(__name__)
app.config['SECRET_KEY'] = '<SECRET KEY>'
sids = []
rooms = collections.defaultdict(list)
socketio = SocketIO(app, cors_allowed_origins='*', async_mode='eventlet',
ping_timeout=5, ping_interval=5)

@socketio.on('cameraList')
def goruntuyu_webe_gonder(data):

    path = data['path'] # Kamera birincil anahtar değeri
    sid = request.sid
    roomName = f"room-{path}"

    if sid not in sids:
        join_room(roomName, sid)
        sids.append(sid)
        rooms[f"{sid}"].append(roomName)
        while True:
            socketio.sleep(0)
            try:
                emit("websocket_consumer", SHARED_MEMORY[path]) #Paylaşımlı bellek
            except Exception as e:
                pass

# Bağlantı bitti
@socketio.on('disconnect')
def baglanti_koptu():
    sid = request.sid
    if sid in sids:
        room = rooms[f"{sid}"]
        leave_room(room[0], sid, namespace="/")
        sids.remove(sid)

socketio.run(app, host=<URL>, port=<PORT>)
```

Son kullanıcıların web sitesinde kamera kontrol ekranında kullanıcı, görüntüleme yetkisine sahip kameraların herbiri için javascript Socket.IO istemci kütüphanesi kullanılarak Socket.IO sunucusuna bağlantı kurmaktadır.

Web Soket İşeyicisi işlemini tamamladıktan sonra kişilerin görüldüğü kamera ve zaman bilgisinin veri tabanında kayıt altına almak için Veri Tabanı İşleyici kullanılır. Gerçek zamanlı işletilen sürecin, veri tabanı tarafında oluşabilecek yavaşlıklardan etkilenmesini engellemek için kaydedilecek veriler "database" Kafka konusuna gönderildikten sonra işlenir. Bütün kameralar için "database" ortak bir Kafka konusudur. Yani veri tabanına eklenecek bütün kişi logları bu konuda toplanır. Apache Kafka [16] "database" Kafka konusunda bulunan veri, Çizelge 8.6'da belirtilen tüketici kod ile veri tabanına kişi logu kaydedilir. Tüketici kod ayrı bir işlem olarak çalıştırılır. İlk olarak *DatabaseConsumer* sınıfının yapıcı fonksiyonunda PostgreSQL [51] veri tabanı bağlantısı kurulur. Sonrasında *run_database_consumer* fonksiyonu çağrılarak veriler işlenmeye başlanır. Fonksiyonda Apache Kafka sunucusuna bağlanmak için *Consumer* nesnesi oluşturulur. *Consumer* nesnesi, verileri okumak için *subscribe* fonksiyonunu kullanarak "database" konusuna abone olur. Abone olunan konudaki mesajlar okunmaya başlanır. Her mesajdaki kişi log bilgileri, bir listede toplandıktan sonra veri tabanına eklenmesi için *database.insertPersonLog* fonksiyonu çağrılır. Bu işlem sonucunda kişi ve kameranın bilgileri, zaman damgası ile birlikte veri tabanındaki *Kisi_Log* tablosuna eklenmektedir.

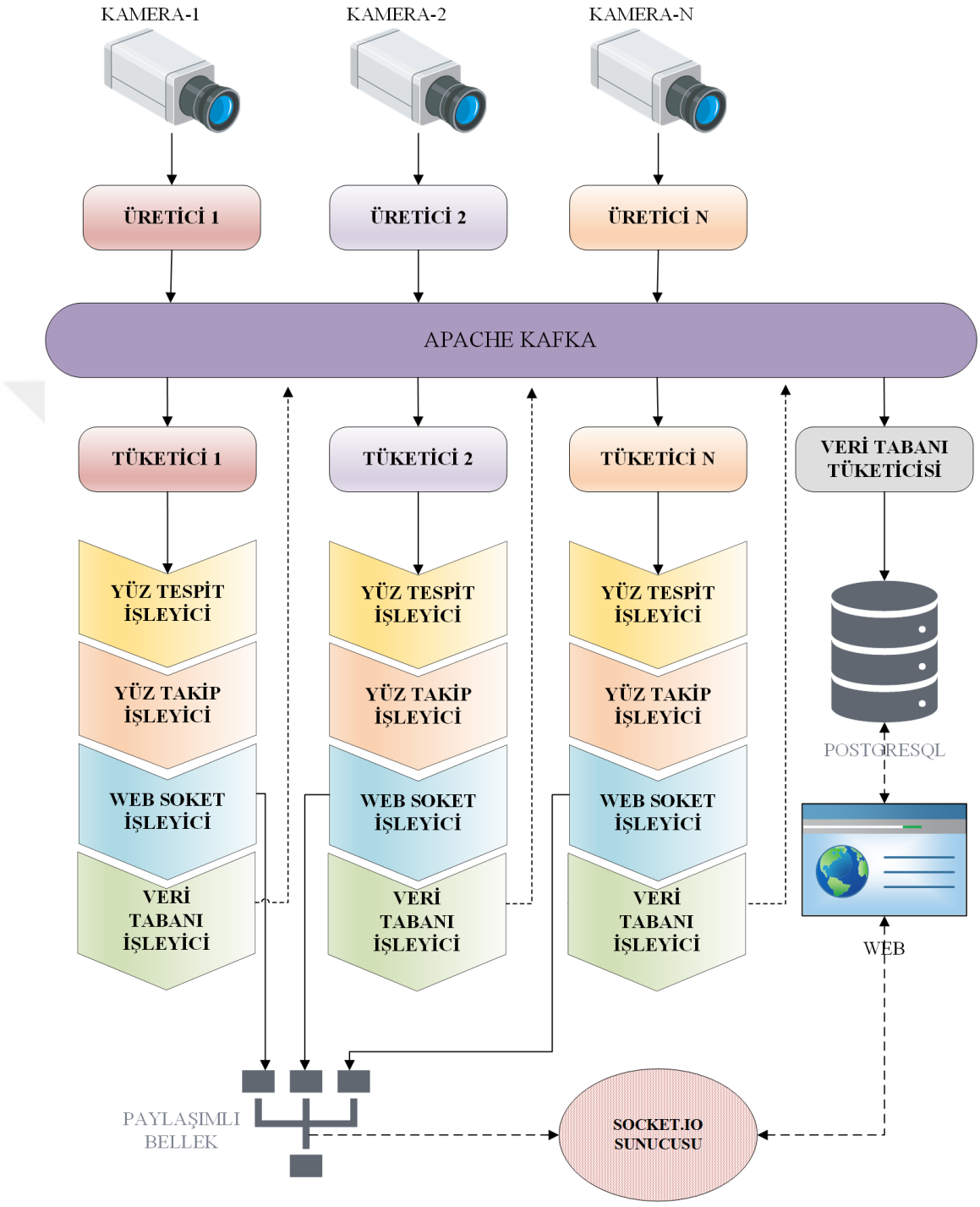
Çizelge 0.12 : Veri tabanı tüketici kod implementasyonu.

```
from typing import Any,List
from confluent_kafka import Consumer
import kafka.consts
from database.database import database
import json
import time
import collections

class DatabaseConsumer():
   def __init__(self):
      self.topic = "database"
      self.consumer = None
      self.database = None
      self.seen_id_list = collections.defaultdict(list)
```

```
    def run_database_consumer(self):

        self.database = database()
        self.database.connect2()

        self.consumer = Consumer({
            'bootstrap.servers': ",".join(kafka.consts.KAFKA_BOOTSTRAP_SERVERS),
            'group.id': 'cameragroup',
            'auto.offset.reset': 'earliest'
        })
        self.consumer.subscribe([self.topic])

        while True:
            msg = self.consumer.poll(1.0)
            if msg is None:
                continue
            if msg.error():
                continue

            arr = json.loads(msg.value())
            person_ids = arr [0]
            camera_id = arr [1]

            if len(person_ids) > 0:

                for person_id in person_ids:
                    self.seen_id_list[camera_id].append([person_id])

                self.database.insertPersonLog(self.seen_id_list,
kafka.consts.MINUMUM_SEEN_FRAME)
                self.seen_id_list = collections.defaultdict(list)
```

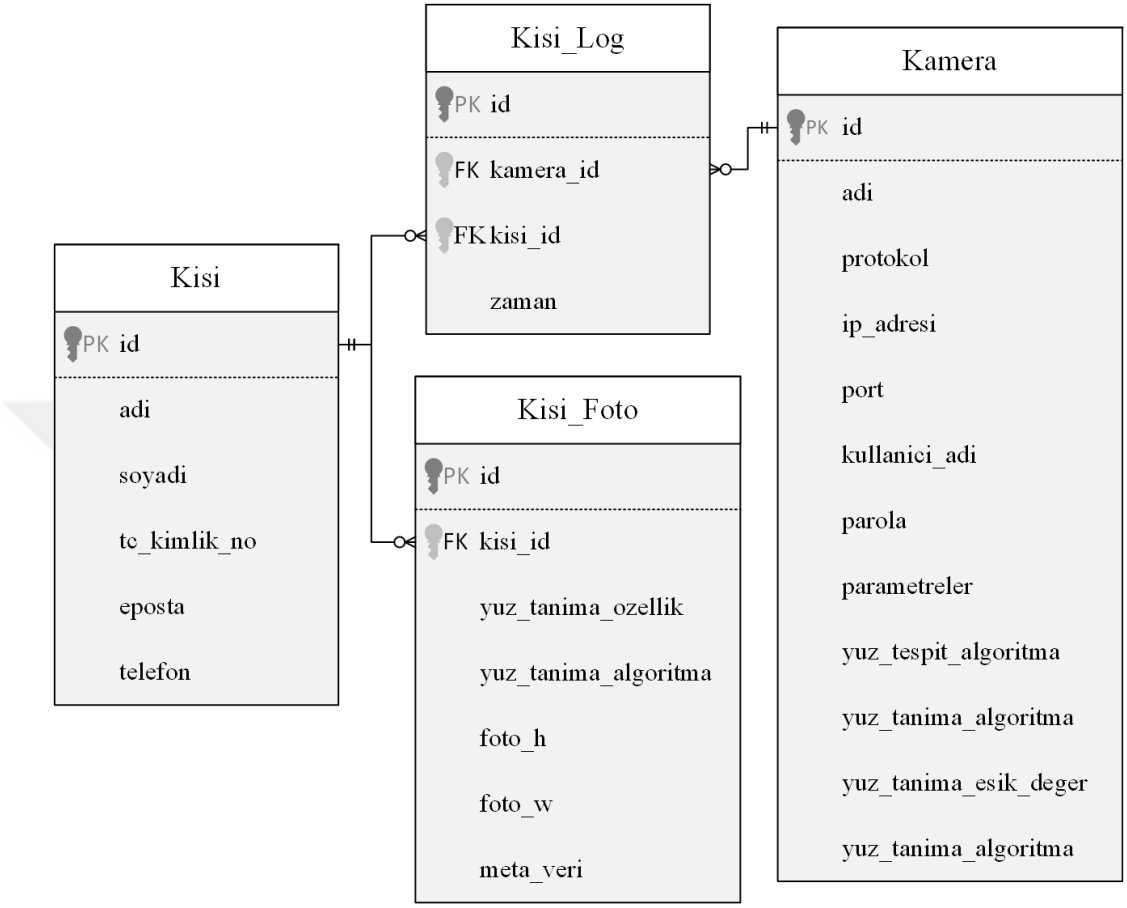
Şekil 8.7'de belirtilen çizim gerçekleştirilen sistem mimarisini ifade edilmektedir. İlk aşamada kameradan görüntünün alınarak web sitesinde son kullanıcıya sunulana kadar kullanılan iletişim ağında görüntü aşamalar arası aktarılırken veri tabanındaki kamera birincil anahtar değeri kullanılmıştır. Sistem içerisinde Python programlama dili kullanılmıştır. Kamera sayısı arttıkça Apache Kafka sayesinde yatay ölçeklenebilir bir

sistem kurulabilir. Ayrıca, görüntüler aynı konu için farklı bölümlerde barındırılarak olası bir veri kaybının önüne geçilebilir.



Şekil 0.15 : Sistem mimarisi.

Çoklu kameralar üzerinden yapılan gerçek zamanlı yüz takip sisteminde verilerin saklanabilmesi için PostgreSQL [51] veri tabanı üzerinde yapılan ilişkisel tasarım Şekil 8.8'de verilmiştir.



Şekil 0.16 : İlişkisel veri tabanı tasarımı.

İlişkisel veri tabanı tasarımında Kisi, Kisi_Foto, Kamera, Kisi_Log tablolarına yer verilmiştir. Aşağıda tablolar ve tablo alanlarının tanımları açıklanmıştır.

- **Kisi**: Kişi bilgilerini saklayan tablodur.
    - id: Kişinin birincil anahtar değerini saklar.
    - adi: Kişinin adını saklar.
    - soyadi: Kişinin soyadını saklar.
    - tc_kimlik_no: Kişinin T.C. kimlik numarasını saklar.
    - eposta: Kişinin eposta bilgisini saklar.
    - telefon: Kişinin telefon bilgisini saklar.
- **Kisi_Foto**: Kişinin farklı yüz tanıma algoritmalarını saklayan tablodur.
    - kisi_id: Kişinin birincil anahtar değerini saklar.
    - yuz_tanima_ozellik: Kişiye ait yüz tanıma algoritmasında çıkarılan özellikleri saklar.
    - yuz_tanima_algoritma: Yüz tanıma özelliklerini çıkarmak için kullanılacak algoritmayı saklar.
    - foto: Kişiye ait fotoğraf bilgisini saklar.
    - foto_w: Fotoğrafın genişlik bilgisini saklar.
    - foto_h: Fotoğrafın uzunluk bilgisini saklar.
    - meta_veri: Fotoğrafın türünü saklar.
- **Kamera**: Kamera bilgilerini saklayan tablodur.

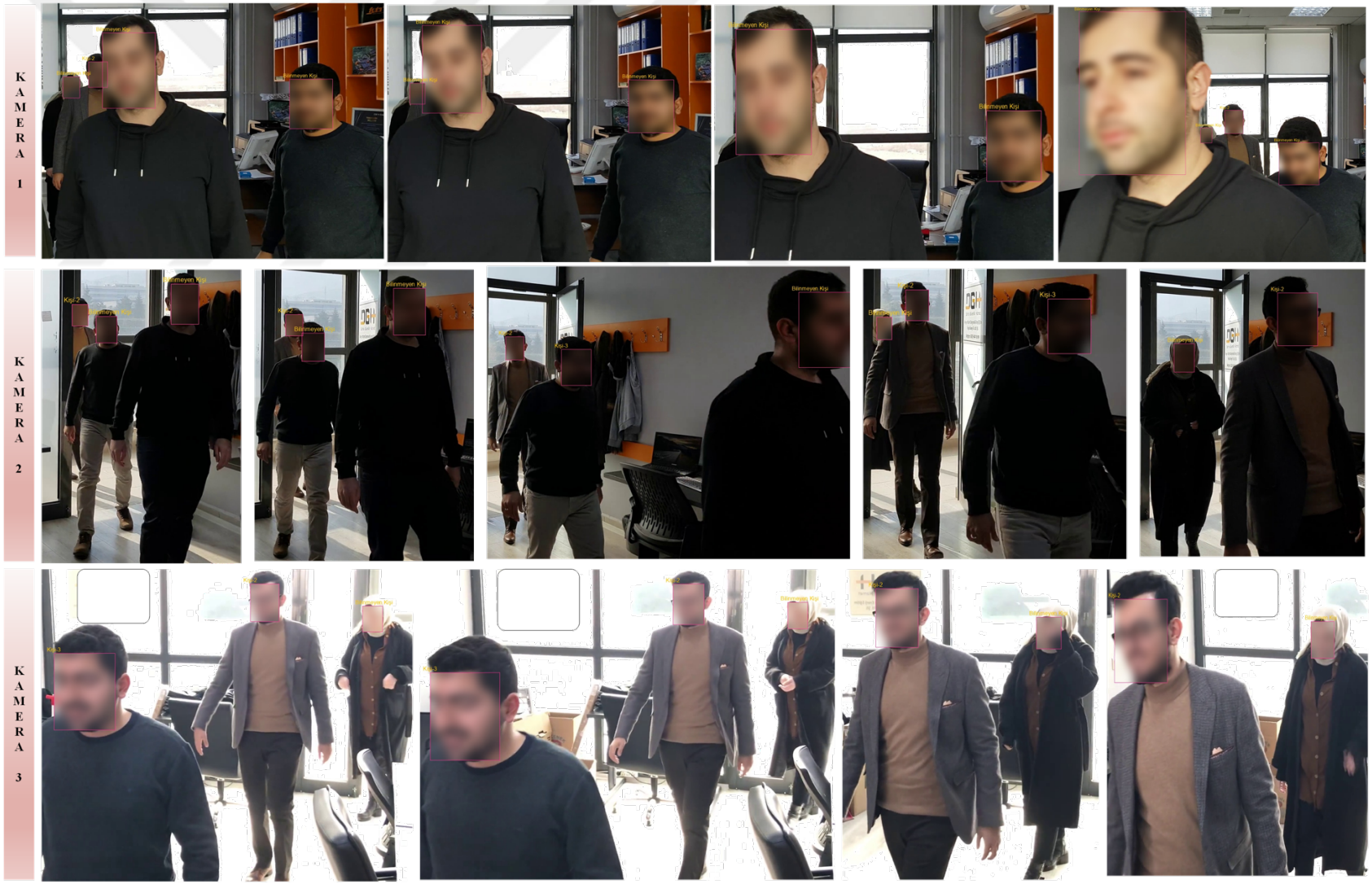
- adi: Kameranın adını saklar.
- protokol: Kameraya erişim sağlanırken kullanılacak protokolleri saklar. Sadece http, https, rtsp protokollerine izin verilmiştir.
- ip_adresi: Kameraya erişim sağlamak için kullanılacak ip adresini saklar.
- port: Kameraya erişim sağlamak için kullanılacak port numarasını saklar.
- kullanici_adi: Kameraya erişiminde kimlik doğrulamasında kullanılacak kullanıcı adını saklar.
- parola: Kameraya erişiminde kimlik doğrulamasında kullanılacak parolayı saklar.
- parametreler: Kameraya erişim linkinin sonundaki varsa parametreleri saklar.
- yuz_tespit_algoritma: Kamera için kullanılacak yüz tespit algoritmasını saklar.
- yuz_tanima_algoritma: Kamera için kullanılacak yüz tanıma algoritmasını saklar.
- yuz_tanima_esik_deger: Yüz tanıma algoritmasında kullanılacak olan eşik değeri saklar.
- aktif: Kameranın aktiflik durumunu saklar.

- **Kisi_Log**: Kameralarda yüzü tespit edilip tanımlanan kişiyi, göründüğü kamera ve zaman bilgisi ile birlikte saklayan tablodur.
  - kisi_id: Kişinin birincil anahtar değerini saklar.

- kamera_id: Kameranın birincil anahtar değerini saklar.

- zaman: Kişinin kamerada göründüğü zamanı saklar.

Çalışma, PyTorch makine öğrenmesi kütüphanesi kullanılarak, AMD Ryzen 5 5600X işlemci, NVIDIA RTX 3060 ekran kartı, 16 GB belleğe sahip bir bilgisayarda yapılmıştır. Şekil 8.9'da test çalışmasında uygulama çıktılarına yer verilmiştir. Test aşamasında farklı konumlarda yer alan toplam 3 kamera kullanılmıştır. Toplam 4 kişinin bulunduğu ortamda kişi yüzleri anonimleştirilmiştir. Veri tabanında Kişi-2 ve Kişi-3 olmak üzere 2 kişiye ait kimlik tanımlanmıştır. Görüntülerde yer alan diğer 2 kişi ise veri tabanında yer almayan yani bilinmeyen kişilerdir. Kamera-1'de Kişi-2 tespit edilebilirken, Kişi-3 tespit edilememiştir. Kamera-2'de Kişi-2 ve Kişi-3 tespit edilmiştir. Kamera-3'te Kişi-2 ve Kişi-3 tespit edilmiştir.



Şekil 0.17 : Uygulamada çoklu kamera görüntüleri.

9. SONUÇ VE ÖNERİLER

Yapılan çalışmalarda belirlenen derin öğrenme yaklaşımları ile çevrimiçi nesne takibi konusunda literatüre katkı sağlanmıştır.

İlk çalışmada FairMOT çoklu nesne takibi algoritmasının ince ayar eğitiminde SGİ, Adam, RMSprop, Rprop optimizasyon algoritmaları kullanılarak karşılaştırma yapılmıştır. Optimizasyon algoritmalarının farklı problemlerde performans metrikleri üzerinde etkisinin olduğu birçok çalışmada vurgulanmaktadır. Bu sebeple FairMOT gibi başarılı bir modeli, optimizasyon algoritmalarının nasıl etkilediği incelenmiştir. Deneysel sonuçlar, RMSprop optimizasyon algoritmasının başarıyı daha fazla arttırdığını göstermiştir. MOT20 doğrulama veri setinde ortalama en yüksek doğruluk değeri, RMSprop optimizasyon algoritması ile %76.7 olarak elde edilmiştir. Gelecekte yeni optimizasyon algoritmaları ile başarı daha fazla arttırılabilir veya başka modeller üzerinde RMSprop'un etkisi araştırılabilir.

İkinci çalışmada ise Çoklu kamera sistemlerinde yüz takibi yapmak istenildiğinde birçok zorluk ile karşılaşılmaktadır. İlk olarak, her kameradan akan görüntünün birbirinden bağımsız bir şekilde ve eş zamanlı işlenmesi gerekmektedir. Aynı zamanda sistem dağıtık ve ölçeklenebilir olarak kurgulanabilmelidir. Yani kamera sayısı arttıkça veya gerek duyulduğu zaman yük birden fazla sunucuya dağıtılabilmelidir. İkinci zorluk ise hata toleranslı bir sistem kurgulanarak oluşabilecek veri kayıplarının önüne geçmektir. Hata toleransı, bir tek verinin bile önemli olduğu yerlerde vazgeçilmez bir unsurdur. Ayrıca, sistem içerisinde yer alan yüz takibinin, kararlı, verimli ve az maliyetli olarak bir arada sunulması zordur. Diğer taraftan sistemde çıktıların zaman damgası ile birlikte depolanabilmesi de önemlidir. Diğer bir zorluk ise son kullanıcı tarafından, işlenen görüntülerin neredeyse gerçek zamanlı olarak izlenebilmesi güvenliğin kritik olduğu yerlerde hayati öneme sahiptir. Bunun için son kullanıcı ile sistem arasında bir iletişim hattının kurgulanarak veri gönderimi gerçek zamanlı olarak yapılmalıdır.

Yapılan çalışmada Yüz tespitinde SCRFD, yüz takibinde ise ArcFace derin öğrenme modelleri kullanılmıştır. Yüz takibi çalışmasında her video akış görüntüsünde tespit edilen

yüzlerin, her defasında yüz tanıma algoritmaları ile ilişkilendirmek yerine geçmişteki verilerden faydalanılarak yüz tanıma işlemi bir kere yapıldıktan sonra yüz takibinin yapılması sağlanmıştır. DeepSORT modelindeki eşleştirmeler için her seferinde yüz tanıma yerine daha küçük evrişimli sinir ağının kullanılması, maliyeti düşürmüştür. Ayrıca DeepSORT tasarımındaki en yakın komşu mesafe algoritmaları, birleşimlerin kesişimi ve Kalman filtresi yaklaşımları ile daha kararlı bir sistem oluşturulmuştur. Böylece yüz kimliği tespit edildikten sonra ilerleyen görüntülerde yüzün tam olarak seçilmediği durumlarda bile kimlik kolay şekilde takip edilebilmektedir. Yukarıda vurgulanan sonuçlara dayanarak yüz takibinin daha kararlı, verimli ve az maliyetli olarak gerçekleştirildiği gösterilmiştir.

Gerçek zamanlı sistem için Apache Kafka akış işleme sistemi ile Socket.IO çift yönlü bağlantı kütüphanesinden faydalanılmıştır. Apache Kafka hataya toleranslı olduğu için oluşabilecek veri kayıplarının önüne geçilmiştir. Kullanılan akış işleme sistemi sayesinde veri tabanı gibi giriş/çıkış işlemlerinin gerçek zamanlı işleyişi aksatmasının önüne geçilmiştir. Her kamera için işletim sistemi seviyesinde ayrı işlemlerin oluşturulması sistem kaynaklarının daha etkili, verimli ve ölçeklenebilir şekilde kullanılmasına olanak sağlamıştır.

İleride yapılacak çalışmalarda farklı yüz tespiti veya yüz tanıma algoritmaları kullanılarak daha az maliyetli veya daha başarılı bir çalışma yapılabilir. Ayrıca kamera sayısının fazla olduğu ortamlarda dağıtık olarak kurgulanmasının etkileri ve başarısı test edilmeye müsaittir. Bunlara ek olarak, DeepSORT yerine farklı nesne takip yaklaşımları ile kararlılık ve verimlilik üzerine bir çalışma yürütülebilir. DeepSORT algoritmasındaki eşleştirme karar araçları optimize edilerek daha kararlı bir sistem kurgulanabilir. DeepSORT algoritmasında yer alan evrişimli sinir ağı güncellenerek daha verimli bir sistem yapılabilir.

KAYNAKLAR

[1] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2016-Decem, pp. 779–788, 2016, doi: 10.1109/CVPR.2016.91.

[2] R. Girshick, "Fast R-CNN," *Proc. IEEE Int. Conf. Comput. Vis.*, vol. 2015 Inter, pp. 1440–1448, 2015, doi: 10.1109/ICCV.2015.169.

[3] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 580–587, 2014, doi: 10.1109/CVPR.2014.81.

[4] M. F. Ozdemir, A. Arı, and D. Hanbay, "Çoklu Nesne Takibi FairMOT Algoritması İçin Optimizasyon Algoritmalarının Karşılaştırılması Comparison of Optimization Algorithms for Multi-Object Tracking FairMOT Algorithm," *Comput. Sci.*, vol. IDAP-2021, no. Special, pp. 147–153, 2021, doi: 10.53070/bbd.990086.

[5] S. Ren, K. He, R. Girshick, and J. Sun, "Faster R-CNN: Towards Real-Time Object Detection with Region Proposal Networks," *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 39, no. 6, pp. 1137–1149, 2017, doi: 10.1109/TPAMI.2016.2577031.

[6] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," *Proc. - 30th IEEE Conf. Comput. Vis. Pattern Recognition, CVPR 2017*, vol. 2017-Janua, pp. 6517–6525, 2017, doi: 10.1109/CVPR.2017.690.

[7] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv*, 2018.

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

[9] N. Wojke, A. Bewley, and D. Paulus, "Simple online and realtime tracking with a deep association metric," *Proc. - Int. Conf. Image Process. ICIP*, vol. 2017-Septe, pp. 3645–3649, 2018, doi: 10.1109/ICIP.2017.8296962.

[10] K. Zhang, Z. Zhang, Z. Li, and Y. Qiao, "Joint Face Detection and Alignment Using Multitask Cascaded Convolutional Networks," *IEEE Signal Process. Lett.*, vol. 23, no. 10, pp. 1499–1503, 2016, doi: 10.1109/LSP.2016.2603342.

[11] J. Deng, J. Guo, E. Ververas, I. Kotsia, and S. Zafeiriou, “Retinaface: Single-shot multi-level face localisation in the wild,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 5202–5211, 2020, doi: 10.1109/CVPR42600.2020.00525.

[12] J. Guo, J. Deng, A. Lattas, and S. Zafeiriou, “Sample and Computation Redistribution for Efficient Face Detection,” 2021, [Online]. Available: http://arxiv.org/abs/2105.04714.

[13] W. Liu, Y. Wen, Z. Yu, M. Li, B. Raj, and L. Song, “SphereFace: Deep Hypersphere Embedding for Face Recognition,” in *2017 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2017, pp. 6738–6746, doi: 10.1109/CVPR.2017.713.

[14] H. Wang *et al.*, “CosFace: Large Margin Cosine Loss for Deep Face Recognition,” in *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2018, pp. 5265–5274, doi: 10.1109/CVPR.2018.00552.

[15] J. Deng, J. Guo, N. Xue, and S. Zafeiriou, “ArcFace: Additive angular margin loss for deep face recognition,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, vol. 2019-June, no. 2, pp. 4685–4694, 2019, doi: 10.1109/CVPR.2019.00482.

[16] “Apache Kafka,” 2021. http://kafka.apache.org.

[17] “Socket.IO,” 2021. https://socket.io/.

[18] C. Szegedy *et al.*, “Going deeper with convolutions,” in *2015 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2015, pp. 1–9, doi: 10.1109/CVPR.2015.7298594.

[19] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, “The PASCAL visual object classes challenge 2007 (VOC2007) Results.” http://www.pascal-network.org/challenges/VOC/voc2007/workshop/index.html, 2007.

[20] T. Y. Lin *et al.*, “Microsoft COCO: Common objects in context,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 8693 LNCS, no. PART 5, pp. 740–755, 2014, doi: 10.1007/978-3-319-10602-1_48.

[21] O. Russakovsky *et al.*, “ImageNet Large Scale Visual Recognition Challenge,” *Int. J. Comput. Vis.*, vol. 115, no. 3, 2015, doi: 10.1007/s11263-015-0816-y.

[22] C. Y. Wang, H. Y. Mark Liao, Y. H. Wu, P. Y. Chen, J. W. Hsieh, and I. H. Yeh, “CSPNet: A new backbone that can enhance learning capability of CNN,” in *IEEE Computer Society Conference on Computer Vision and Pattern Recognition Workshops*, 2020, vol. 2020-June, doi: 10.1109/CVPRW50498.2020.00203.

[23] K. He, X. Zhang, S. Ren, and J. Sun, “Spatial Pyramid Pooling in Deep Convolutional Networks for Visual Recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, vol. 37, no. 9, pp. 1904–1916, Sep. 2015, doi: 10.1109/TPAMI.2015.2389824.

[24] T. Y. Lin, P. Dollár, R. Girshick, K. He, B. Hariharan, and S. Belongie, “Feature pyramid networks for object detection,” in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017, vol. 2017-January, doi: 10.1109/CVPR.2017.106.

[25] S. Liu, L. Qi, H. Qin, J. Shi, and J. Jia, “Path Aggregation Network for Instance Segmentation,” 2018, doi: 10.1109/CVPR.2018.00913.

[26] C. Cortes and V. Vapnik, “Support-vector networks,” *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995, doi: 10.1007/BF00994018.

[27] N. E. C. L. America and P. Nj, “Large-Scale Machine Learning with Stochastic Gradient Descent (SGD),” *Proc. COMPSTAT’2010*, pp. 3–4, 2010, doi: 10.1007/978-3-7908-2604-3.

[28] X. Zhou, D. Wang, and P. Krähenbühl, “Objects as points,” *arXiv*, 2019.

[29] Y. Zhang, C. Wang, X. Wang, W. Zeng, and W. Liu, “FairMOT: On the Fairness of Detection and Re-Identification in Multiple Object Tracking,” pp. 1–13, 2020, [Online]. Available: http://arxiv.org/abs/2004.01888.

[30] Y. Zhang *et al.*, “ByteTrack: Multi-Object Tracking by Associating Every Detection Box,” 2021, [Online]. Available: http://arxiv.org/abs/2110.06864.

[31] A. Bewley, Z. Ge, L. Ott, F. Ramos, and B. Upcroft, “Simple online and realtime tracking,” *Proc. - Int. Conf. Image Process. ICIP*, vol. 2016-Augus, pp. 3464–3468, 2016, doi: 10.1109/ICIP.2016.7533003.

[32] F. Yu, D. Wang, E. Shelhamer, and T. Darrell, “Deep Layer Aggregation,” *Proc. IEEE Comput. Soc. Conf. Comput. Vis. Pattern Recognit.*, pp. 2403–2412, 2018, doi: 10.1109/CVPR.2018.00255.

[33] A. Milan, L. Leal-Taixe, I. Reid, S. Roth, and K. Schindler, “MOT16: A Benchmark for Multi-Object Tracking,” pp. 1–12, 2016, [Online]. Available: http://arxiv.org/abs/1603.00831.

[34] Z. Ge, S. Liu, F. Wang, Z. Li, and J. Sun, “YOLOX: Exceeding YOLO Series in 2021,” pp. 1–7, 2021, [Online]. Available: http://arxiv.org/abs/2107.08430.

[35] L. Chen, H. Ai, Z. Zhuang, and C. Shang, “Real-time multiple people tracking with deeply learned candidate selection and person re-identification,” *arXiv*, 2018.

[36] K. Bernardin and R. Stiefelhagen, “Evaluating multiple object tracking performance: The CLEAR MOT metrics,” *Eurasip J. Image Video Process.*, vol. 2008, 2008, doi: 10.1155/2008/246309.

[37] E. Ristani, F. Solera, R. Zou, R. Cucchiara, and C. Tomasi, “Performance measures and a data set for multi-target, multi-camera tracking,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 9914 LNCS, no. c, pp. 17–35, 2016, doi: 10.1007/978-3-319-48881-3_2.

[38] J. Luiten *et al.*, “HOTA: A Higher Order Metric for Evaluating Multi-object Tracking,” *Int. J. Comput. Vis.*, vol. 129, no. 2, pp. 548–578, 2021, doi: 10.1007/s11263-020-01375-2.

[39] M. Riedmiller and H. Braun, “Direct adaptive method for faster backpropagation learning: The RPROP algorithm,” *1993 IEEE Int. Conf. Neural Networks*, pp. 586–591, 1993, doi: 10.1109/icnn.1993.298623.

[40] T. Tieleman, G. Hinton, and others, “RMSProp,” *COURSERA Neural networks Mach. Learn.*, vol. 4, no. 2, pp. 26–31, 2012.

[41] D. P. Kingma and J. L. Ba, “Adam: A method for stochastic optimization,” *3rd Int. Conf. Learn. Represent. ICLR 2015 - Conf. Track Proc.*, pp. 1–15, 2015.

[42] S. Yang, P. Luo, C.-C. Loy, and X. Tang, “WIDER FACE: A Face Detection Benchmark,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Jun. 2016, pp. 5525–5533.

[43] G. B. Huang, M. Mattar, T. Berg, and E. Learned-Miller, “Labeled faces in the wild: A database forstudying face recognition in unconstrained environments,” 2008.

[44] M. Köstinger, P. Wohlhart, P. M. Roth, and H. Bischof, “Annotated Facial Landmarks in the Wild: A large-scale, real-world database for facial landmark localization,” in *2011 IEEE International Conference on Computer Vision Workshops (ICCV Workshops)*, 2011, pp. 2144–2151, doi: 10.1109/ICCVW.2011.6130513.

[45] V. Jain and E. Learned-Miller, “FDDB: A Benchmark for Face Detection in Unconstrained Settings,” 2010.

[46] Y. Zhu, H. Cai, S. Zhang, C. Wang, and Y. Xiong, “Tinaface: Strong but simple baseline for face detection,” *arXiv Prepr. arXiv2011.13183*, 2020.

[47] D. Yi, Z. Lei, S. Liao, and S. Z. Li, “Learning face representation from scratch,” *arXiv Prepr. arXiv1411.7923*, 2014.

[48] L. Wolf, T. Hassner, and I. Maoz, “Face recognition in unconstrained videos with matched background similarity,” in *CVPR 2011*, 2011, pp. 529–534, doi: 10.1109/CVPR.2011.5995566.

[49] I. Kemelmacher-Shlizerman, S. M. Seitz, D. Miller, and E. Brossard, “The MegaFace Benchmark: 1 Million Faces for Recognition at Scale,” in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, 2016, pp. 4873–4882, doi: 10.1109/CVPR.2016.527.

[50] P. Dendorfer *et al.*, “MOT20: A benchmark for multi object tracking in crowded scenes,” *arXiv*, pp. 1–7, 2020.

[51] M. Stonebraker and L. A. Rowe, “The Design of POSTGRES,” *ACM SIGMOD Rec.*, vol. 15, no. 2, 1986, doi: 10.1145/16856.16888.

[52] “OpenCV,” 2021. https://opencv.org/.

[53] J. Deng, J. Guo, J. Yang, N. Xue, I. Cotsia, and S. P. Zafeiriou, “ArcFace: Additive Angular Margin Loss for Deep Face Recognition,” *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 1, 2021, doi: 10.1109/TPAMI.2021.3087709.

ÖZGEÇMİŞ

Ad-Soyad : Mehmet Fatih ÖZDEMİR

ÖĞRENİM DURUMU:

- Lisans: 2005 – 2010, Hacettepe Üniversitesi, Mühendislik Fakültesi, Bilgisayar Mühendisliği Bölümü

MESLEKİ DENEYİM:

- 2020 – Devam, İnönü Üniversitesi, Bilgisayar Mühendisliği Bölümünde Araştırma Görevlisi
- 2019 – 2020, Forte Teknoloji, Lider Yazılım Uzmanı
- 2018 – 2019, Dışişleri Bakanlığı, Sözleşmeli Bilişim Personeli
- 2016 – 2018, Devlet Su İşleri Genel Müdürlüğü, Sözleşmeli Bilişim Personeli
- 2012 – 2016, Erbul Bilgi Teknolojileri, Kıdemli Yazılım Uzmanı
- 2010 – 2011, Probase Bilgi Teknolojileri, Yazılım Uzmanı

YÜKSEK LİSANS TEZİNDEN TÜRETİLEN ÇALIŞMALAR

- M. F. Ozdemir, A. Arı, and D. Hanbay, "Çoklu Nesne Takibi FairMOT Algoritması İçin Optimizasyon Algoritmalarının Karşılaştırılması Comparison of Optimization Algorithms for Multi-Object Tracking FairMOT Algorithm," *Comput. Sci.*, vol. IDAP-2021, no. Special, pp. 147–153, 2021, doi: 10.53070/bbd.990086.