## ИНФОРМАТИКА

**ӘОЖ 004.8**
**МҒТАР 28.23**
**МҒТАР 14.35.09**
https://doi.org/10.47526/2024-3/2524-0080.10

**A.N. AMANOV[1], A. B. ABEN[2]**
[1]*PhD, Hoca Ahmet Yesevi Türk-Kazak Üniversitesi,*
*(Kazakistan, Türkistan), e-mail: anuarbek.amanov@ayu.edu.kz*
[2]*Doctora Öğrencisi, Hoca Ahmet Yesevi Türk-Kazak Üniversitesi*
*(Kazakistan, Türkistan), e-mail: arypzhan.aben@ayu.edu.kz*

# GÖRÜNTÜ VERİLERİNDE NESNE TESPİTİ VE SINIFLANDIRILMASINDA KULLANILAN ÇEŞİTLİ ALGORİTMALARIN İNCELENMESİ VE UYGULAMASI

**Özet.** Bu makalede, dijital görüntü işleme ve nesne tespiti alanında yaygın olarak kullanılan YOLO (You Only Look Once - Sadece Bir Kez Bakarsınız) yöntemi ile OpenCV kütüphanesi kullanılarak uygulanan Haar özniteliklerine dayalı kaskad sınıflandırıcı yöntemi karşılaştırmalı olarak incelenmiştir. YOLO, derin öğrenmeye dayalı bir yöntem olup, özellikle gerçek zamanlı nesne algılama ve tanıma uygulamalarında öne çıkmaktadır. Haar yöntemi ise geleneksel bir yaklaşımla hızlı bir şekilde özellik tanımlama yeteneğine sahiptir. Ancak, bu iki yöntem arasında performans açısından önemli farklar bulunmaktadır. Deneyler ve performans analizleri, YOLO'nun nesne algılama görevlerinde daha yüksek doğruluk oranları ve gerçek zamanlı işlem hızı sunduğunu ortaya koymuştur. Bu çalışmada kullanılan yöntemler ve kodlar, dijital görüntü işlemeye yeni başlayan araştırmacılara önemli katkılar sağlayabilir. Ayrıca, YOLO'nun GPU desteğinden yararlanarak büyük ve karmaşık veri kümelerinde yüksek performans sağladığı gösterilmiştir. YOLO'nun farklı sürümleri (örneğin, YOLOv4, YOLOv5, YOLOv7) ile yapılan deneyler, düşük gecikme süresi ve yüksek doğruluk performansı nedeniyle gerçek zamanlı uygulamalar için en uygun sürümlerden bazıları olduğunu ortaya koymuştur.

**Anahtar kelimeler:** *Görüntü işleme, Nesne tanıma, OpenCV, YOLO,* ***Haar Yöntemi***

**А.Н. Аманов[1], А.Б. Абен[2]**
[1]*PhD, аға оқытушы*
*Қожа Ахмет Ясауи атындағы Халықаралық қазақ-түрік университеті,*
*(Қазақстан, Түркістан қ.), e-mail: anuarbek.amanov@ayu.edu.kz*
[2]*Докторант, Қожа Ахмет Ясауи атындағы Халықаралық қазақ-түрік университеті,*
*(Қазақстан, Түркістан қ). e-mail: arypzhan.aben@ayu.edu.kz*

# Кескін деректеріндегі объектілерді анықтау және жіктеу үшін қолданылатын алгоритмдерді зерттеу және қолдану

**Андатпа.** Бұл мақалада цифрлық кескінді өңдеу арқылы объектілерді анықтау саласында кеңінен қолданылатын YOLO (You Only Look Once) әдісі мен OpenCV кітапханасы арқылы жүзеге асырылған Haar мүмкіндіктеріне негізделген каскадты жіктеуіш әдісінің көрсеткіштері салыстырылып зерттеледі. YOLO - терең оқытуға негізделген әдіс және әсіресе нақты уақыттағы нысанды анықтау және тану қолданбаларында ерекшеленеді. Хаар әдісі, керісінше, дәстүрлі тәсілмен белгілі бір мүмкіндіктерге негізделген жылдам

анықтау мүмкіндігіне ие. Дегенмен, бұл екі әдіс арасында айтарлықтай өнімділік айырмашылықтары бар. Тәжірибелер мен өнімділікті талдау нәтижесінде YOLO нысанды анықтау тапсырмаларында жоғары дәлдік жылдамдықтары мен нақты уақыттағы өңдеу жылдамдығын ұсынатыны анықталды. Зерттеудегі қолданылған әдістердегі кодтар цифрлық кескінді өңдеу бағытында жаңадан зертеушілер үшін үлкен көмегі тиеді. YOLO үлкен және күрделі деректер жиындарында GPU мүмкіндітерін қолдануымен жұмыс істеу арқылы жоғары тиімділікті қамтамасыз ететіндігі көрсетілген. Сонымен қатар, YOLO-ның әртүрлі нұсқаларымен жүргізілген эксперименттер (мысалы, YOLOv4, YOLOv5, YOLOv7) бұл нақты уақыттағы қолданбалар үшін ең қолайлы нұсқалардың бірі екенін көрсетті, әсіресе оның төмен кідіріс және жоғары дәлдік көрсеткіштері арқасында.

**Түйін сөздер:** *Кескінді цифрлық өңдеу, Обьект тану, OpenCV, YOLO,* ***Haar әдісі***

**А.Н. Аманов[1], А.Б. Абен[2]**

[1] *PhD, старший преподаватель, Международного казахско-турецкого университета имени Ходжи Ахмеда Ясави (Казахстан, г. Туркестан), e-mail: anuarbek.amanov@ayu.edu.kz*

[2]*Доктиорант, Международного казахско-турецкого университета имени Ходжи Ахмеда Ясави (Казахстан, г. Туркестан), e-mail: arypzhan.aben@ayu.edu.kz*

**Изучение и применение различных алгоритмов для обнаружения и классификации объектов на изображениях**

**Аннотация.** В данной статье сравниваются и изучаются метод YOLO (You Only Look Once), широко используемый в области обнаружения объектов посредством цифровой обработки изображений, и метод каскадного классификатора на основе функций Хаара, реализованный с помощью библиотеки OpenCV. YOLO — это метод, основанный на глубоком обучении, который особенно эффективен в приложениях для обнаружения и распознавания объектов в реальном времени. С другой стороны, метод Хаара позволяет быстро идентифицировать признаки, опираясь на традиционный подход. Однако между этими двумя методами существуют значительные различия в производительности. Эксперименты и анализ производительности показали, что YOLO обеспечивает высокую точность и скорость обработки в реальном времени при решении задач обнаружения объектов. Программный код, использованный в исследовании, может оказать большую помощь исследователям, которые мало знакомы с цифровой обработкой изображений. Было продемонстрировано, что YOLO достигает высокой производительности при работе с большими и сложными наборами данных благодаря использованию вычислительных возможностей графического процессора. Кроме того, эксперименты с различными версиями YOLO (например, YOLOv4, YOLOv5, YOLOv7) показали, что версия YOLOv7 является одной из наиболее подходящих для приложений в реальном времени, благодаря низкой задержке и высокой точности.

**Ключевые слова:** *Цифровая обработка изображений, Распознавание объектов, OpenCV, YOLO, метод Haar*

**A.N. Amanov[1], A.B. Aben[2]**

[1]*PhD, Senior Lecturer of Khoja Akhmet Yassawi International Kazakh-Turkish University, (Kazakhstan, Turkistan), e-mail: anuarbek.amanov@ayu.edu.kz*

[2]*Graduate PhD student of Khoja Akhmet Yassawi International Kazakh-Turkish University (Kazakhstan, Turkistan), e-mail: arypzhan.aben@ayu.edu.kz*

**Investigation and Application of Various Algorithms used in Object Detection and**

**Classification in Image Data**

**Abstract.** In this article, we compare and analyze the YOLO (You Only Look Once) method, widely employed for object detection in digital image processing, with the Haar feature-based cascade classifier method implemented using the OpenCV library. YOLO, a deep learning–based approach, excels in real-time object detection and recognition applications. In contrast, the Haar method utilizes a traditional approach to rapidly identify features. However, significant performance differences exist between the two methods. Experimental results and performance analyses demonstrate that YOLO provides high accuracy rates and real-time processing speeds in object detection tasks. The code implementations presented in this study will be valuable to researchers new to digital image processing. Additionally, YOLO has shown high performance on large and complex datasets by leveraging GPU capabilities. Experiments with various YOLO versions (e.g., YOLOv4, YOLOv5, YOLOv7) have established it as one of the most suitable options for real-time applications, particularly due to its low latency and high accuracy.

**Keywords:** *Digital image processing, object recognition, OpenCV, YOLO, Haar method*

**Giriş**

Son yıllarda bilgisayarlı görme ve nesne tanıma teknolojileri, otonom araçlar, video gözetim sistemleri, tıbbi teşhis, perakende satış, tarım ve daha birçok alanda devrim yaratmıştır. Nesne tanıma teknolojisi, sistemlerin görüntülerdeki ve videolardaki çeşitli nesneleri otomatik olarak tanımlamasını ve sınıflandırmasını sağlar. Yapay zeka ve makine öğrenmesindeki hızlı gelişmelerle birlikte, nesne tanıma, araştırma ve uygulama açısından en aktif ve dinamik alanlardan biri haline gelmiştir.

Geleneksel yöntemler, Destek Vektör Makineleri (SVM) ve Histogram of Oriented Gradients (HOG) gibi tekniklere dayanarak nesneleri tanımaya çalışırken, bu yöntemler yerini daha ileri tekniklere bırakmıştır. Özellikle Evrişimli Sinir Ağları (CNN), Bölgesel Evrişimli Sinir Ağları (R-CNN), YOLO (You Only Look Once) ve Mask R-CNN gibi derin öğrenme yöntemleri, nesne tanımada devrim yaratmıştır. Bu modern yaklaşımlar, yüksek doğruluk ve güvenilirlik sağlarken, gerçek zamanlı çözümlemelere de imkan tanır.

YOLO gibi yöntemler, görüntüyü tek bir ileri beslemeli sinir ağı ile analiz ederek yüksek hız ve doğruluk sunar. Bu özellik, özellikle gerçek zamanlı uygulamalar için çok değerlidir. OpenCV ve TensorFlow gibi popüler kütüphaneler, nesne tanıma algoritmalarının uygulanmasını ve geliştirilmesini kolaylaştırır. Bu kütüphaneler, araştırmacıların ve geliştiricilerin projelerinde hızlı ilerleme kaydetmelerine olanak tanır.

Nesne tanıma, tıbbi görüntüleme gibi alanlarda da önemli bir rol oynar. Örneğin, bilgisayarlı tomografi (CT) taramaları ve manyetik rezonans görüntüleme (MRI) gibi tıbbi görüntülerde tümörlerin ve anormalliklerin otomatik tespiti, teşhis sürecini hızlandırabilir ve doğruluğunu artırabilir. Tarımda, hastalıklı bitkilerin ve zararlıların erken tespiti, verimliliği artırabilir ve maliyetleri azaltabilir.

Perakende sektöründe, raflarda bulunan ürünlerin otomatik tanınması ve envanter yönetimi süreçlerinin iyileştirilmesi müşteri memnuniyetini artırabilir ve operasyonel verimliliği sağlayabilir. Otonom araçlar için nesne tanıma, çevredeki araçları, yayaları ve diğer nesneleri tanıyarak güvenli sürüş sağlar. Video gözetim sistemlerinde ise, nesne tanıma, güvenlik ihlallerini hızlı bir şekilde tespit ederek güvenlik seviyesini artırır.

Görüntü tanıma ve nesne tespitinde bilgisayarlı görü algoritmalarının uygulanması, bu algoritmaların görüntülerdeki nesneleri ve sahneleri doğru şekilde tanımlayıp sınıflandırmalarına odaklanan önemli bir alandır. Yapılan çalışmalarda algoritmalar %89 ile %97 arasında bir doğruluk oranına ulaşmaktadır [1]. Bu çalışma ayrıca, görüntü sınıflandırmasında yumuşak hesaplama tekniklerinin rolünü inceleyerek ortaya çıkan uygulamaları ve kullanılan çeşitli sınıflandırma

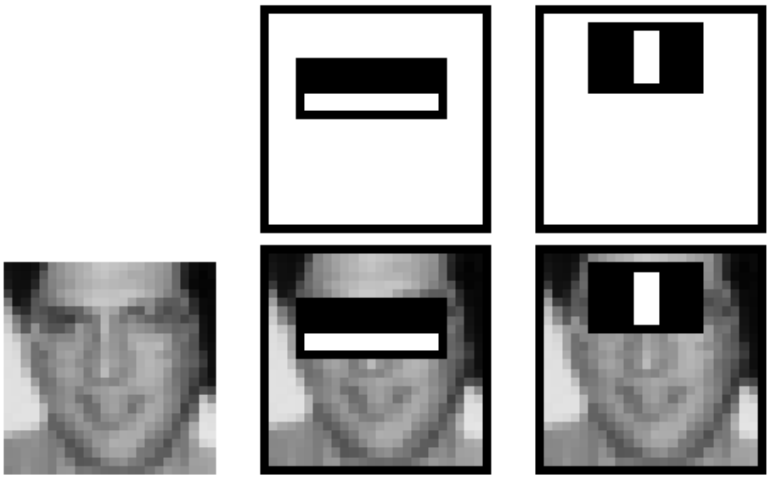
yöntemleri hakkında bilgi sunmaktadır [2]. Çeşitli metodolojilerin güçlü ve zayıf yönlerinin kapsamlı bir analizi yapılarak, pratik uygulamalarda bilinçli karar almayı kolaylaştırmaktadır [3].

Makine öğrenimi algoritmaları, desen tanıma teknikleri kullanarak yüksek boyutlu ve çok modlu biyomedikal verileri analiz etmekte, bu sayede hastalık teşhisinin doğruluğunu artırmakta ve tıbbi görüntülemede tarafsız karar vermeye destek olmaktadır [4]. Makine öğrenimi, derin öğrenme, evrişimli sinir ağları (CNN), transfer öğrenmesi ve diğer ileri seviye görüntü işleme teknolojileri, çalışmada kapsamlı bir şekilde ele alınmış ve özetlenmiştir [5]. Makale, CNN, SVM, ANN ve lojistik regresyon gibi görüntü sınıflandırma algoritmalarının karşılaştırmalı analizini sunarak, CNN'in diğer modellerden sürekli olarak daha iyi performans gösterdiğini, SVM, ANN ve lojistik regresyonun ise performans sıralamasında onu takip ettiğini ortaya koymaktadır [6].

**İlgili çalişmalar**

Son yıllarda, nesne tespiti ve tanıma alanındaki araştırmalar, derin öğrenme tabanlı yaklaşımların, geleneksel yöntemlerle karşılaştırılmasına odaklanmıştır. YOLO (You Only Look Once) yöntemi ve Haar özelliği tabanlı kaskad sınıflandırıcı gibi yöntemler, bu alandaki temel teknolojiler arasında yer almaktadır. 2020 yılından bu yana, birçok çalışma bu yöntemlerin performanslarını çeşitli veri setleri ve senaryolar üzerinde karşılaştırmıştır[7].

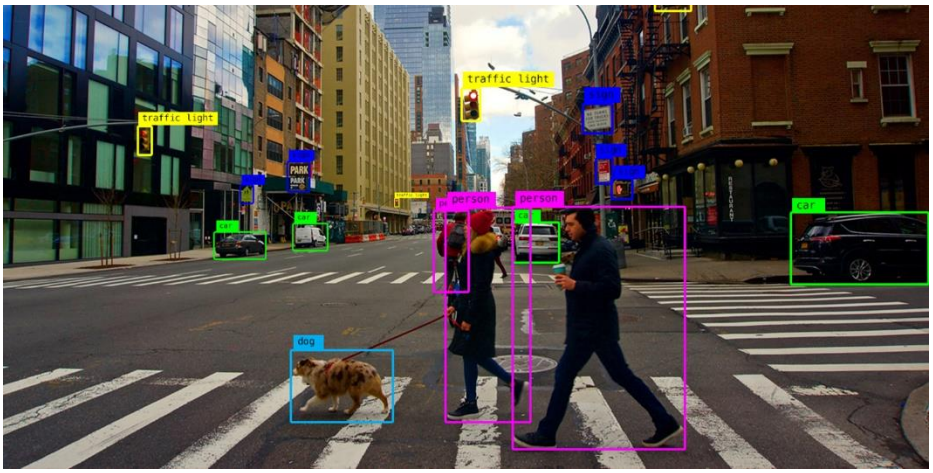
***A. YOLO Yöntemine İlişkin Çalışmalar.*** YOLO yöntemi, son yıllarda nesne tespiti alanında en çok kullanılan yöntemlerden biri olmuştur. YOLOv4 ve YOLOv5'in performanslarını çeşitli veri kümelerinde karşılaştırarak YOLOv4'ün daha hızlı ve daha doğru sonuçlar verdiğini, ancak YOLOv5'in daha hafif bir mimari ile daha düşük hesaplama maliyetleri sunduğunu göstermiştir[8]. YOLOv7'nin, karmaşık sahnelerdeki nesne tespitinde hem hız hem de doğruluk açısından üstün performans sağladığını ve gerçek zamanlı uygulamalar için en uygun seçeneklerden biri olduğunu belirtmiştir[9].



***Şekil 2.1.*** *Resimlerle OpenCV'de Haar kademesinin çalışması*

***B. OpenCV ve Haar Yöntemine İlişkin Çalışmalar.*** OpenCV'nin Haar özelliği tabanlı kaskad sınıflandırıcı yöntemi, nesne tespitinde geleneksel bir yaklaşım olarak kullanılmaya devam etmektedir (Şekil 2.1). Haar yönteminin düşük donanım kapasitesine sahip cihazlar için hala geçerli bir çözüm sunduğunu, ancak derin öğrenme tabanlı yöntemlerle karşılaştırıldığında doğruluk açısından geride kaldığını vurgulamıştır[10]. Haar kaskad yönteminin, yüz ve göz gibi belirli nesnelerin tespitinde hala etkili olduğunu, ancak

genel nesne tespiti görevlerinde derin öğrenme yaklaşımlarının gerisinde kaldığını göstermiştir.




***Şekil 2.2.*** *Sıfırdan YOLO Tiny Özelleşmiş Nesne Tespiti*

***Karşılaştırmalı Çalışmalar:*** YOLO ve Haar yöntemlerinin performanslarını karşılaştıran birkaç çalışma, her iki yöntemin de belirli kullanım durumlarına göre avantajları olduğunu ortaya koymuştur. YOLO ve Haar yöntemlerini çeşitli video veri kümeleri üzerinde karşılaştırmış ve YOLO'nun daha yüksek doğruluk ve hız sunduğunu, Haar yönteminin ise daha düşük donanım maliyetleriyle daha basit uygulamalar için uygun olduğunu belirtmiştir. YOLO ve Haar yöntemlerinin farklı uygulama alanlarındaki başarılarını incelemiş ve YOLO'nun, özellikle büyük ölçekli ve gerçek zamanlı uygulamalarda üstün performans gösterdiğini vurgulamıştır (Şekil 2.2) [11].

Bu çalışmalar, YOLO ve Haar yöntemlerinin farklı koşullarda nasıl performans gösterdiğini ve hangi uygulama senaryolarında en uygun olduklarını anlamak için kapsamlı bir temel oluşturmaktadır.

Ayrıca, büyük ölçekli eğitim verilerinin eksikliğinin üstesinden gelmek için stratejiler de incelenmiştir. Veri artırma, transfer öğrenmesi, meta öğrenme gibi yöntemler ile ilgili veri kümeleri, değerlendirme kriterleri ve yöntem karşılaştırmaları tartışılmıştır [12]. Çalışma ayrıca, özellikle YOLO serisi, SSD ve RetinaNet gibi derin öğrenme tabanlı nesne algılama algoritmalarını inceleyerek, bu algoritmaların evrimi, teknik detayları, karşılaştırmalı performansları, uygulamaları ve görüntü veri analizi konusundaki sınırlamalarına dair içgörüler sunmaktadır [13]. Ek olarak, nesne tespiti için sinirsel yaklaşımlar gözden geçirilmiş, STARE, DRIVE ve ImageNet gibi veri kümeleri kullanılarak Naïve Bayes ve LMS gibi makine öğrenimi modellerinin doğruluğu karşılaştırılmıştır [14].

Görüntü sınıflandırması için basit bir CNN modeli oluşturulmuş ve sınıflandırma performansını en fazla etkileyen temel parametreleri belirlemek amacıyla farklı öğrenme oranları ve optimizasyon algoritmaları kullanılarak test edilmiştir [15]. Bu inceleme, YOLO algoritmasının kronolojik gelişimini izleyerek, yıllar içerisindeki önemli ilerlemelerini ve iyileştirmelerini vurgulamaktadır. Ayrıca, nesne tespiti teknolojilerinin büyük potansiyeli ve çok yönlülüğü gözler önüne serilmiştir [16]. Makale, aşırı yumuşatma gibi sorunlara çözüm bulmak amacıyla, görüntü sınıflandırmasında grafik evrişimli ağların (GCN) rolünü incelemektedir. GCN'ler, grafik yapılandırılmış verileri işlemekte yüksek performans sergileyerek görüntü analizinde gelecekteki yeniliklere kapı aralamaktadır [17].

Çalışma, görüntü sınıflandırmasında CNN'ler kullanılarak Gabor, LBP ve SIFT filtreleri ile aykırı değer algılama tekniklerini karşılaştırmakta ve geleneksel yöntemlere kıyasla uzatılmış eğitim süreleriyle birlikte daha yüksek doğrulukta anormallik algılama sağlamaktadır [18]. Ayrıca,

sokak seviyesinde nesne algılamada kullanılan YOLOv5, YOLOv4, YOLOv3 ve SSD MobileNetv2 FPN-lite gibi tek aşamalı dedektör algoritmalarının karşılaştırmalı bir analizini sunmakta, bu algoritmalar arasında YOLOv5l'nin en doğru ve verimli sonuçları verdiği belirtilmektedir [19]. Son olarak, sokak seviyesinde algılama görevlerinde YOLOv5l'nin üstün performansı vurgulanmıştır [20].

**Nesne tanıma algoritmalarının uygulanması**

*Haar yönteminin uygulanması*

Google Colab'da Haar yöntemini kullanacak kod. Kodu çalıştırmak için Haar Cascade Sınıflandırıcısını indirin (Şekil 3.1).

```
[2] !wget https://github.com/opencv/opencv/raw/master/data/haarcascades/haarcascade_frontalface_default.xml

--2024-05-20 05:52:24--  https://github.com/opencv/opencv/raw/master/data/haarcascades/haarcascade_frontalface_default.xml
Resolving github.com (github.com)... 140.82.116.4
Connecting to github.com (github.com)|140.82.116.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml [following]
--2024-05-20 05:52:24--  https://raw.githubusercontent.com/opencv/opencv/master/data/haarcascades/haarcascade_frontalface_default.xml
Resolving raw.githubusercontent.com (raw.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to raw.githubusercontent.com (raw.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 930127 (908K) [text/plain]
Saving to: 'haarcascade_frontalface_default.xml'

haarcascade_frontal 100%[===================>] 908.33K  --.-KB/s    in 0.04s

2024-05-20 05:52:24 (24.0 MB/s) - 'haarcascade_frontalface_default.xml' saved [930127/930127]
```

***Şekil 3.1.*** *Haar basamaklı sınıflandırıcıyı yükleme*

Haar basamaklı sınıflandırıcıyı yükledikten sonra kodu yazıp çalıştırıyoruz. Bu kod, Google Colab ortamındaki OpenCV kitaplığını kullanarak Haar basamaklı sınıflandırıcıyı kullanan yüzleri tanımak için tasarlanmıştır. Kod görüntüyü okur, gri tonlamalı hale getirir, yüzleri tanır ve tanınan yüzleri bir dikdörtgenle işaretler ve ardından sonucu görüntüler (Şekil 3.2).

```python
import cv2
from google.colab.patches import cv2_imshow


face_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_frontalface_default.xml')


image = cv2.imread('/content/Без названия.jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)


faces = face_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))


for (x, y, w, h) in faces:
    cv2.rectangle(image, (x, y), (x+w, y+h), (255, 0, 0), 2)

cv2_imshow(image)
```

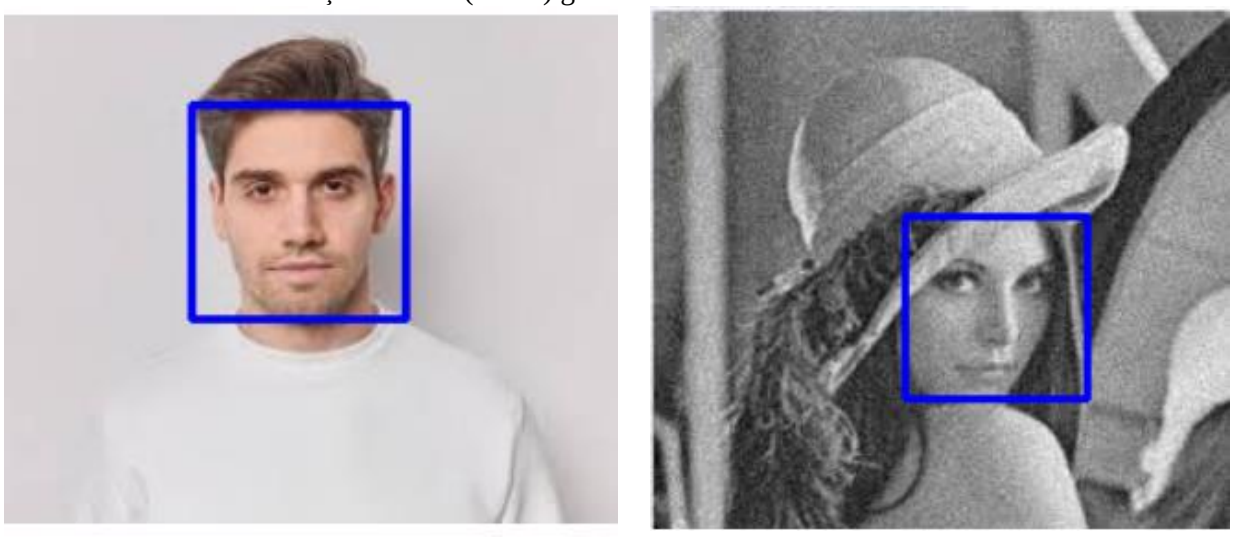
***Şekil 3.2.*** *Haar yöntemine kullanılan kod*

Kodu çalıştırma adımları:

1. *OpenCV kütüphanesinin kurulumu: OpenCV kütüphanesini kodunuzda kullanmak için kurmanız gerekmektedir.*
2. *Haar Cascade Sınıflandırıcıyı İndirin: Yüz tanıma için gerekli Haar Cascade dosyasını indirin.*
3. *Resim yükle: Yüklenen resmi okuyun.*
4. *Gri tonlamalı: Daha kolay tanınması için görüntüyü gri tonlamalı hale getirir.*
5. *Yüz algılama: DetectMultiScale işlevini kullanarak yüz algılama.*
6. *Sayfaları işaretle: Tanınan sayfaları bir dikdörtgenle işaretleyin.*

*7. Sonucu göster: Sonucu göstermek için cv2_imshow işlevini kullanın.*

Bu kodun sonucu Şekil 3.3'te (a ve b) gösterilmektedir.



a) b)

***Şekil 3.3.*** *Kodu çalıştırdıktan sonraki sonuç*

İkinci kod, Haar basamaklı sınıflandırıcı kullanılarak oluşturulur. Bu kod, resimdeki araç plakalarını tanımak için tasarlanmıştır. Kod, görüntüyü okur, gri tonlamaya dönüştürür, plakaları algılar ve tanınan plakaları bir dikdörtgenle işaretleyerek sonucu görüntüler (Şekil 3.4).

```
import cv2
import matplotlib.pyplot as plt

plate_cascade = cv2.CascadeClassifier(cv2.data.haarcascades + 'haarcascade_russian_plate_number.xml')

image = cv2.imread('/content/images (1).jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

plates = plate_cascade.detectMultiScale(gray, scaleFactor=1.1, minNeighbors=5, minSize=(30, 30))

for (x, y, w, h) in plates:
    cv2.rectangle(image, (x, y), (x + w, y + h), (255, 0, 0), 2)


plt.imshow(image)
plt.axis('off')
plt.show()
```

***Şekil 3.4.*** *Haar yöntemine kullanılan kod*

Kodu çalıştırma adımları:

1. *Sınıflandırıcıyı cv2.CascadeClassifier('haarcascade_russian_plate_number.xml') kullanarak yüklüyoruz.*
2. *Resmi cv2.imread(image_path) kullanarak okuyoruz.*
3. *cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) kullanarak görüntüyü griye çeviriyoruz.*
4. *DetectMultiScale fonksiyonunu kullanarak sayı sembollerini tanıyoruz.*
5. *Tanınan sayı sembollerini bir kare ile işaretliyoruz ve sonucu plt.imshow kullanarak gösteriyoruz.*

Bu adımlarda aşağıdaki sonucu alıyoruz (***Şekil 3.5***):


***Şekil 3.5.*** *Kodu çalıştırdıktan sonraki sonuç*

**YOLO (You Only Look Once) yönteminin uygulanması**

Google Colab'da YOLO yöntemini kullanacak kod. Önceden ayarlanmış ve konfigürasyon dosyasın indirme yolu Şekil 3.6'da gösterilmektedir.

```
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.cfg
!wget https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
```

```
--2024-05-20 10:34:05--  https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.cfg
Resolving github.com (github.com)... 140.82.113.4
Connecting to github.com (github.com)|140.82.113.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/c6725600-96cf-11ea-9a3b-5f6c6e35d23b?X-Am
--2024-05-20 10:34:05--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/c6725600-96cf-11ea-9a3b-5f
Resolving objects.githubusercontent.com (objects.githubusercontent.com)... 185.199.108.133, 185.199.109.133, 185.199.110.133, ...
Connecting to objects.githubusercontent.com (objects.githubusercontent.com)|185.199.108.133|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 13351 (13K) [application/octet-stream]
Saving to: 'yolov4.cfg'

yolov4.cfg          100%[==================>]  13.04K  --.-KB/s    in 0s

2024-05-20 10:34:06 (37.2 MB/s) - 'yolov4.cfg' saved [13351/13351]

--2024-05-20 10:34:06--  https://github.com/AlexeyAB/darknet/releases/download/darknet_yolo_v3_optimal/yolov4.weights
Resolving github.com (github.com)... 140.82.114.4
Connecting to github.com (github.com)|140.82.114.4|:443... connected.
HTTP request sent, awaiting response... 302 Found
Location: https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/ba4b6380-889c-11ea-9751-f994f5961796?X-Am
--2024-05-20 10:34:06--  https://objects.githubusercontent.com/github-production-release-asset-2e65be/75388965/ba4b6380-889c-11ea-9751-f9
```

***Şekil 3.6.*** *Yapılandırma dosyasının ve YOLO ağırlıklarının yüklenmesi*

Bu kod, bir görüntüdeki nesneleri algılamak için YOLO (Yalnızca Bir Kez Bakarsınız) modelini kullanmanızı sağlar. Gerekli ağırlık ve model konfigürasyon dosyalarını[21] otomatik olarak yükler ve ardından bunları, görüntülenen görüntüdeki nesneler hakkında tahminlerde bulunmak için kullanır (Şekil 3.7).

```
import cv2
import matplotlib.pyplot as plt
import numpy as np

# Загрузка изображения
image_path = '/content/slices-ripe-red-watermelon-isolated-spruce-background_271580-25.jpg' # Замените 'example.jpg' на путь к вашему изображению
image = cv2.imread(image_path)

# Функция для обнаружения объектов на изображении
def detect_objects(image):
    net = cv2.dnn.readNet('yolov4.weights', 'yolov4.cfg')
    layers_names = net.getLayerNames()
    output_layers = [layers_names[i - 1] for i in net.getUnconnectedOutLayers()]


    blob = cv2.dnn.blobFromImage(image, 0.00392, (416, 416), (0, 0, 0), True, crop=False)
    net.setInput(blob)
    outs = net.forward(output_layers)

    class_ids = []
    confidences = []
    boxes = []
    for out in outs:
        for detection in out:
            scores = detection[5:]
            class_id = np.argmax(scores)
            confidence = scores[class_id]
            if confidence > 0.5:
                center_x = int(detection[0] * image.shape[1])
                center_y = int(detection[1] * image.shape[0])
                w = int(detection[2] * image.shape[1])
                h = int(detection[3] * image.shape[0])
                x = int(center_x - w / 2)
                y = int(center_y - h / 2)
                boxes.append([x, y, w, h])
                confidences.append(float(confidence))
                class_ids.append(class_id)

    indexes = cv2.dnn.NMSBoxes(boxes, confidences, 0.5, 0.4)
    font = cv2.FONT_HERSHEY_PLAIN
    colors = np.random.uniform(0, 255, size=(len(class_ids), 2))

    for i in range(len(boxes)):
        if i in indexes:
            x, y, w, h = boxes[i]
            label = str(class_ids[i])
            color = colors[i]
            cv2.rectangle(image, (x, y), (x + w, y + h), color, 2)
            cv2.putText(image, label, (x, y + 30), font, 3, color, 2)

    plt.figure(figsize=(10, 6))
    plt.imshow(cv2.cvtColor(image, cv2.COLOR_BGR2RGB))
    plt.axis('off')
    plt.show()

# Обнаружение объектов на изображении
detect_objects(image)
```

***Şekil 3.7.*** YOLO yöntemini kullanacak kod

Kodu çalıştırma adımları:

*1. Gerekli kütüphaneleri içe aktarın*

*2. Bir resim yükleyin*

*3. Nesne algılama fonksiyonunu tanımlayın cv2.dnn.readNet('yolov4.weights', 'yolov4.cfg'): Ağırlıklar ve yapılandırma dosyasıyla birlikte önceden oluşturulmuş YOLO modeli yüklendi.*

*net.getLayerNames(): ağın tüm katmanlarının adlarını alır.*

*net.getUnconnectedOutLayers(): Diğer katmanlara bağlı olmayan ağ çıkış katmanlarının indekslerini alır. Bu indeksler çıktı katmanlarının adlarını almak için kullanılır.*

*4. Görüntü hazırlama ve tahmin cv2.dnn.blobFromImage(...): Bir görüntüyü sinir ağına (blob) beslemeye uygun bir formata dönüştürür.net.setInput(blob): Oluşturulan görüntüyü network.net.forward(output_layers) için giriş olarak ayarlar): Tahminleri elde etmek için ağ üzerinden ileterek geçişi gerçekleştirin.*

*5. Tahmin sonuçlarının işlenmesi class_ids Sınıf kimliklerini ( ), güvenleri ( güvenleri ) ve dikdörtgen koordinatları ( ) tutacak başlatılmış kutuları listeler. Her tahmin koordinatları, boyutları, güvenilirlikleri ve sınıf tanımlayıcılarını içerir. np.argmax(scores): En yüksek güvene sahip sınıfı belirleyin. güven > 0,5: Tahminleri güven sınırına göre filtreleyin. Tanımlanan nesnelerin etrafındaki dikdörtgenlerin koordinatları ve boyutları hesaplanır. Koordinatlar, güvenler ve sınıf kimlikleri ilgili listelere eklenir.*

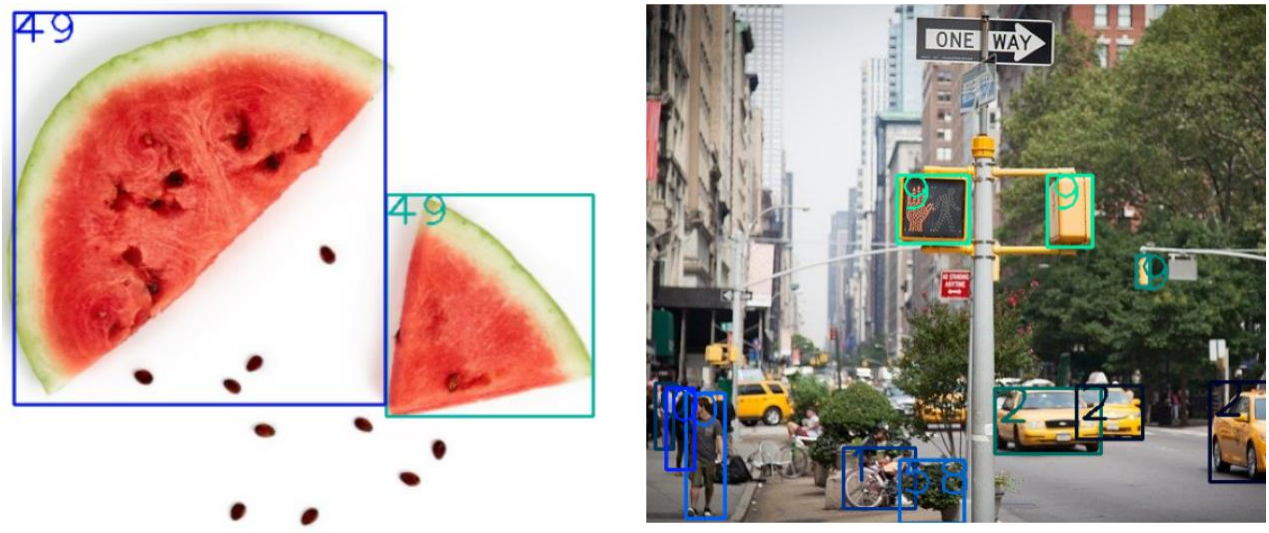
*6. Maksimum olmayan sıkıştırma kullanın ve dikdörtgenler*

*cv2.dnn.NMSBoxes(...): Üst üste binen dikdörtgenleri kaldırmak için maksimum olmayan sıkıştırma algoritması uygulayın. Farklı sınıflar için rastgele renkler oluşturun. Maksimum olmayan değerlere tıkladıktan sonra kalan tüm dikdörtgenleri yineleyin. Resmin üzerine dikdörtgenler ve sınıf işaretleri çizin.*

*7. Sonucu görüntüleme*

*8. Çağrı işlevi*

Bu kodun sonucu Şekil 3.8’de (a ve b) gösterilmektedir.




a) b)

***Şekil 3.8.*** *Kodu çalıştırdıktan sonraki sonuç*

**OpenCV kütüphanesinin kullanımı**

Google Colab'da OpenCV kütüphanesinin kullanacak kod. Bu kod, belirli bir görüntüdeki nesneleri algılamak ve saymak için OpenCV kitaplığını kullanır[22,23]. Kod, bir görüntüyü okur, onu gri tonlamaya dönüştürür ve ardından konturları algılamak için Canney'in algoritmasını kullanır. Tanımlanan konturların alanı belirli bir değerden büyükse nesne olarak kabul edilirler (Şekil 3.9).

```
import cv2
from google.colab.patches import cv2_imshow

image = cv2.imread('/content/Без названия (1).jpg')
gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
edged = cv2.Canny(gray, 30, 150)

contours, _ = cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE)

objects = 0
for contour in contours:
  area= cv2.contourArea(contour)
  if area > 50:
    objects += 1
    cv2.drawContours (image, [contour], -1, (0, 0, 255), 2)

print ("Nesne Sayısı:", objects)
cv2_imshow(image)
```

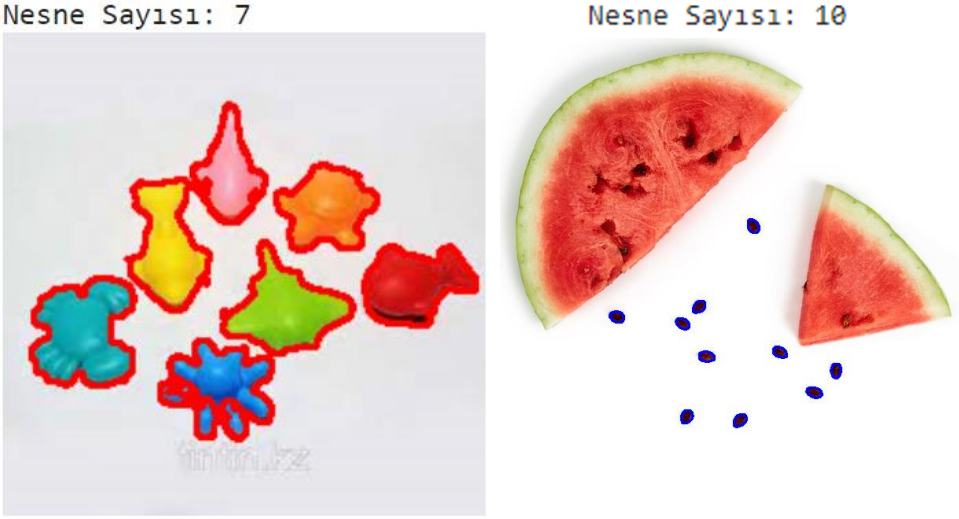
***Şekil 3.9.*** *OpenCV kütüphanesi ile kullanılan kod*

Kodu çalıştırma adımları:

1. *OpenCV ve Google Colab araçlarını içe aktarın.*
2. *Resmin okunması ve gri tonlamaya dönüştürülmesi: cv2.imread kullanarak resmi okuyoruz. cv2.cvtColor(image, cv2.COLOR_BGR2GRAY) kullanarak görüntüyü griye çeviriyoruz[10].*
3. *Konturların belirlenmesi: cv2.Canny(gray, 30, 150) kullanarak Canny algoritmasını kullanarak görüntünün kenarlarını belirleriz. cv2.findContours(edged.copy(), cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) kullanarak konturları buluruz[11].*
4. *Konturlarla çalışın: nesneler = 0 başlangıçta nesnelerin sayısı sıfırdır. Konturlardaki kontur için: her konturu kontrol ederiz. Kontur alanını cv2.contourArea(contour)*

*kullanarak hesaplıyoruz. Alan 50'den büyükse nesne sayısını arttırıp konturunu resim üzerinde işaretliyoruz.*

5. *Sonucu görüntüleyin: print("Nesne sayısı:", nesneler) bulunan nesnelerin sayısını gösterir. cv2_imshow(image) bir resmi gösterir[12].*
6. Bu kodun sonucu Şekil 3.10'da (a ve b) gösterilmektedir.




a) b)

***Şekil 3.10.*** *Kodu çalıştırdıktan sonraki sonuç*

**Sonuç**

Bu çalışmada, YOLO (You Only Look Once) yöntemi ile OpenCV kütüphanesi kullanılarak Haar özelliği tabanlı kaskad sınıflandırıcı yönteminin performansları karşılaştırılmıştır. Araştırma üniversite lisans üstü örencilerine ve yeni araştırmacılar için de ek yardımcı olacaktır. Kullanılan her iki yöntem de nesne tespiti ve tanıma görevlerinde yaygın olarak kullanılsa da elde edilen sonuçlar, YOLO'nun modern derin öğrenme tabanlı yaklaşımı sayesinde Haar yöntemine göre belirgin avantajlar sunduğunu göstermektedir.

Performans açısından, YOLO yöntemi daha yüksek doğruluk ve düşük yanlış pozitif oranları ile öne çıkmaktadır. YOLO, gerçek zamanlı nesne tespitinde hız açısından büyük bir avantaj sağlarken, Haar yöntemi özellikle eski ve düşük donanım gereksinimi olan sistemler için tercih edilebilir. Ancak, büyük ve karmaşık veri setlerinde ya da yüksek çözünürlüklü görüntülerde Haar yöntemi, YOLO'ya kıyasla performans olarak geride kalmaktadır.

Hesaplama maliyetine gelince, YOLO'nun GPU optimizasyonları sayesinde paralel hesaplamalarla büyük veri kümelerinde önemli ölçüde zaman tasarrufu sağladığı görülmektedir. Buna karşılık, Haar yöntemi CPU tabanlı daha basit bir yapıya sahip olduğundan düşük maliyetli cihazlarda dahi kabul edilebilir sonuçlar üretebilir, ancak daha karmaşık senaryolarda performans düşüşü yaşanabilir.

YOLO yöntemi, yüksek doğruluk ve hız gerektiren gerçek zamanlı uygulamalar için daha uygun görünürken, Haar yöntemi donanım kısıtlamaları olan ve daha düşük hassasiyet gerektiren durumlar için alternatif bir çözüm sunmaktadır. Bu bulgular, nesne tespiti uygulamalarında kullanılacak yöntemin seçiminin, kullanım senaryosuna ve donanım kapasitesine göre yapılmasının önemini vurgulamaktadır.

Gelecekteki çalışmalar kapsamında, YOLO'nun farklı sürümleri (örneğin, YOLOv5 veya YOLOv7) ile diğer modern derin öğrenme tabanlı yaklaşımlar karşılaştırılabilir ve farklı veri kümeleri üzerinde bu yöntemlerin performans analizleri daha kapsamlı şekilde genişletilebilir.

## KAYNAKLAR

1. Herui Wang. Application of Computer Vision Algorithms in Image Recognition and Object Detection. Academic Journal of Computing & Information Science (2024), Vol. 7, Issue 1: 59-64. https://doi.org/10.25236/AJCIS.2024.070109.

2. Preeti, Sharma., Rajeev, Kamal, Sharma., Isha, Kansal., Rajeev, Kumar., Rana, Gill. An Extensive Review on Image Classification Techniques for Expert Systems. (11 Dec 2023). doi: 10.2174/0123520965282357231123093259

3. Archana, R., Jeevaraj, P.S.E. Deep learning models for digital image processing: a review. Artif Intell Rev 57, 11 (2024). https://doi.org/10.1007/s10462-023-10631-z

4. Rahul Kumar Dwivedi, Bhanu Prakash, Md Sheesh, Gulrez Akhter, Kartik Meghwal,"ADVANCEMENTS IN DEEP LEARNING OBJECT DETECTION: A COMPREHENSIVE RESEARCH REVIEW ", Futuristic Trends in Artificial Intelligence Volume 3 Book 8,IIP Series, Volume 3, May, 2024, Page no.142-152, e-ISBN: 978-93-6252-759-2, DOI/Link: https://www.doi.org/10.58532/V3BGAI8P2CH6

5. Juan Li, Pan Jiang, Qing An, Gai-Ge Wang, Hua-Feng Kong. Medical image identification methods: A review, Computers in Biology and Medicine, Volume 169, 2024, 107777, ISSN 0010-4825, https://doi.org/10.1016/j.compbiomed.2023.107777.

6. Sneha, K., Verma. 6. Comparative Analysis of Image Classification Algorithms. International Journal for Research in Applied Science and Engineering Technology, (2023). doi: 10.22214/ijraset.2023.57662

7. Burger, W., & Burge, M. J. (2022). *Digital image processing: An algorithmic introduction*. Springer Nature.

8. Petrou, M. M., & Kamata, S. I. (2021). *Image processing: dealing with texture*. John Wiley & Sons.

9. Bailey, D. G. (2023). *Design for embedded image processing on FPGAs*. John Wiley & Sons.

10. Van der Velden, B. H., Kuijf, H. J., Gilhuijs, K. G., & Viergever, M. A. (2022). Explainable artificial intelligence (XAI) in deep learning-based medical image analysis. *Medical Image Analysis*, *79*, 102470.

11. Chen, H., Wang, Y., Guo, T., Xu, C., Deng, Y., Liu, Z., ... & Gao, W. (2021). Pre-trained image processing transformer. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 12299-12310).

12. Qiao, Qi., Azlin, Ahmad., Wang, Ke. 7. Image classification based on few-shot learning algorithms: a review. Indonesian Journal of Electrical Engineering and Computer Science, (2024). doi: 10.11591/ijeecs.v35.i2.pp933-943

13. D., K., Gupta. A Review: Object Detection Algorithms. (2023). doi: 10.1109/ICSCCC58608.2023.10176865

14. Vandna Bhalla. A Review on Neural Approaches in Image Processing Applications. International Journal For Science Technology And Engineering, (2023). doi: 10.22214/ijraset.2023.49851

15. Mandeep, Kaur. A Review on Classification of Images with Convolutional Neural Networks. International Journal For Science Technology And Engineering, (2023). doi: 10.22214/ijraset.2023.54704

16. M. Letavay, M. Bažant and P. Tuček, "Object Detection Algorithms - A Review," 2023 International Conference on Control, Artificial Intelligence, Robotics & Optimization (ICCAIRO), Crete, Greece, 2023, pp. 31-44, doi: 10.1109/ICCAIRO58903.2023.00014.

17. Tang W. Review of Image Classification Algorithms Based on Graph Convolutional Networks. EAI Endorsed Trans AI Robotics [Internet]. 2023 Jul. 6 [cited 2024 Sep. 8];2. Available from: https://publications.eai.eu/index.php/airo/article/view/3462

18. D. J. Dsouza and A. P. Rodrigues, "A Comparative Study of Feature Extraction Methods in Image Classification Using Convolution Neural Network Model," 2023 International Conference on Recent

Advances in Information Technology for Sustainable Development (ICRAIS), Manipal, India, 2023, pp. 77-82, doi: 10.1109/ICRAIS59684.2023.10367096.

19. Martinus, Grady, Naftali., Jason, Sebastian, Sulistyawan., Kelvin, Julian. 14. Comparison of Object Detection Algorithms for Street-level Objects. arXiv.org, (2022). doi: 10.48550/arXiv.2208.11315

20. Xijun, Liang., ShengHao, DU., Yuze, Duan., Yuelin, Chen., Kaili, Zhu., Yitong, Yin., Ling, Jian. 16. Kernel-based Algorithms for Image Classification: A Review. (2023). doi: 10.21203/rs.3.rs-3576956/v1

21. Ngugi, L. C., Abelwahab, M., & Abo-Zahhad, M. (2021). Recent advances in image processing techniques for automated leaf pest and disease recognition–A review. *Information processing in agriculture*, *8*(1), 27-51.

22. Kim, G., Kwon, T., & Ye, J. C. (2022). Diffusionclip: Text-guided diffusion models for robust image manipulation. In *Proceedings of the IEEE/CVF conference on computer vision and pattern recognition* (pp. 2426-2435).

23. Tov, O., Alaluf, Y., Nitzan, Y., Patashnik, O., & Cohen-Or, D. (2021). Designing an encoder for stylegan image manipulation. *ACM Transactions on Graphics (TOG)*, *40*(4), 1-14.