T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ROBOT HÜCRESİ İÇERİSİNDE YAPAY ZEKÂ VE GÖRÜNTÜ İŞLEME TABANLI PARÇA BESLEME KONTROLÜ

YÜKSEK LİSANS TEZİ

Enesalp ÖZ

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Bilim Dalı

HAZİRAN 2025

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

ROBOT HÜCRESİ İÇERİSİNDE YAPAY ZEKÂ VE GÖRÜNTÜ İŞLEME TABANLI PARÇA BESLEME KONTROLÜ

YÜKSEK LİSANS TEZİ

Enesalp ÖZ

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Elektronik Mühendisliği Bilim Dalı

Tez Danışmanı: Doç.Dr. Muhammed Kürşat UÇAR

HAZİRAN 2025

Enesalp ÖZ tarafından hazırlanan “Robot hücresi içerisinde yapay zeka ve görüntü işleme tabanlı parça besleme kontrolü” adlı tez çalışması 17.06.2025 tarihinde aşağıdaki jüri tarafından oy birliği ile Sakarya Üniversitesi Fen Bilimleri Enstitüsü Elektrik-Elektronik Mühendisliği Anabilim Dalı **Elektronik Mühendisliği** Bilim Dalı’nda Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Jürisi

Jüri Başkanı :	**Dr. Öğr. Üyesi M. Ali PALA** Sakarya Uygulamalı Bilimler Üniversitesi	.....
Jüri Üyesi :	**Doç. Dr. M. Kürşad UÇAR** (Danışman) Sakarya Üniversitesi	.....
Jüri Üyesi :	**Doç. Dr. Selçuk EMİROĞLU** Sakarya Üniversitesi	.....

ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

Sakarya Üniversitesi Fen Bilimleri Enstitüsü Lisansüstü Eğitim-Öğretim Yönetmeliğine ve Yükseköğretim Kurumları Bilimsel Araştırma ve Yayın Etiği Yönergesine uygun olarak hazırlamış olduğum "Robot Hücresi İçerisinde Yapay Zeka ve Görüntü İşleme Tabanlı Parça Besleme Kontrolü" başlıklı tezin bana ait, özgün bir çalışma olduğunu; çalışmamın tüm aşamalarında yukarıda belirtilen yönetmelik ve yönergeye uygun davrandığımı, tezin içerdiği yenilik ve sonuçları başka bir yerden almadığımı, tezde kullandığım eserleri usulüne göre kaynak olarak gösterdiğimi, bu tezi başka bir bilim kuruluna akademik amaç ve unvan almak amacıyla vermediğimi ve 20.04.2016 tarihli Resmi Gazete'de yayımlanan Lisansüstü Eğitim ve Öğretim Yönetmeliğinin 9/2 ve 22/2 maddeleri gereğince Sakarya Üniversitesi'nin abonesi olduğu intihal yazılım programı kullanılarak Enstitü tarafından belirlenmiş ölçütlere uygun rapor alındığını, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun ortaya çıkması halinde doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi beyan ederim.

(18/06/2025).

ENESALP ÖZ

Her zaman yanımda olan, desteğini hiçbir zaman esirgemeyen sevgili eşime ve kıymetli anne, babama ithaf ediyorum.

TEŞEKKÜR

Bu çalışmada rehberliği ve desteğiyle bana yol gösteren hocama teşekkür ederim. Katkıları, çalışmamın başarısında önemli bir rol oynamıştır.

Ayrıca, bu projeyi gerçekleştirme sürecinde sağladığı imkânlar ve destek için TMMT'ye teşekkür ederim. Çalışma ortamı, deneyim kazanma fırsatları ve sağlanan teknik olanaklar, bu araştırmanın gelişmesine önemli katkılar sağlamıştır.

Beni her zaman destekleyen eşime ve aileme de sonsuz teşekkür ederim.

Enesalp ÖZ

İÇİNDEKİLER

Sayfa

KISALTMALAR

AI : Yapay Zeka
ANN : Yapay Sinir Ağı
CMOS : Tamamlayıcı Metal Oksit Yarı İletken
DI : Dijital Giriş
DO : Dijital Çıkış
FPS : Saniyedeki kare sayısı
GUI : Grafiksel Kullanıcı Arayüzü
mAP : Ortalama Ortalama Doğruluk
PLC : Programlanabilir mantık denetleyici
RTU : Uzaktan terminal birimi
SVM : Destek Vektör Makineleri (Karmaşık sınıflandırma algoritması)
TCP : Veri iletim protokolü
USB : Universal Serial Bus — Evrensel seri veri yolu
YOLO : You Only Look Once (Algoritmanın ismi)

TABLO LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

ROBOT HÜCRESİ İÇERİSİNDE YAPAY ZEKA VE GÖRÜNTÜ İŞLEME TABANLI PARÇA BESLEME KONTROLÜ

ÖZET

Bu çalışma, endüstriyel üretim hatlarında sıklıkla karşılaşılan parça besleme hatalarını önlemek amacıyla, yapay zekâ tabanlı bir görüntü işleme sistemi geliştirmeyi hedeflemektedir. Günümüz üretim süreçlerinde özellikle otomotiv sektörü gibi yüksek hassasiyet gerektiren alanlarda, robotik sistemlerin doğru çalışması büyük önem taşımaktadır. Bu sistemlerdeki en kritik adımlardan biri olan parça besleme sürecinde oluşabilecek en küçük hata bile, kalite sorunlarına, üretim kayıplarına ve ciddi maliyet artışlarına neden olabilmektedir. Bu kapsamda geliştirilen sistem, operatör tarafından parçaların yerleştirildiği bölgeyi kamera ile görüntüleyerek, parçanın doğru konumda olup olmadığını gerçek zamanlı olarak tespit etmektedir.Geleneksel görüntü işleme yöntemlerinin, özellikle aydınlatma değişimi, çapak gibi fiziksel engeller ve benzer arka plan koşullarında yetersiz kaldığı bilinmektedir. Bu nedenle çalışmada klasik yöntemler yerine, derin öğrenme tabanlı bir yaklaşım tercih edilmiştir. Özellikle evrişimli sinir ağı (CNN) mimarisi üzerine kurulu olan YOLOv7-tiny modeli kullanılarak, hızlı ve doğru bir sınıflandırma sistemi oluşturulmuştur. YOLO mimarisi, görüntüyü bir bütün olarak analiz ederek nesne tespitini hem hızlı hem de yüksek doğrulukla gerçekleştirebilmesi sayesinde bu alanda önemli bir avantaj sunmaktadır.Model eğitimi, üretim sahasından toplanan 2400 adet görüntü ile gerçekleştirilmiştir. Verilerin %50'si doğru yerleştirilmiş (OK), %50'si ise hatalı yerleştirilmiş (NG) parçaları içermektedir. NG sınıfındaki verilerin azlığı, yapay veri üretimi ve veri artırma teknikleri ile dengelenmiştir. Bu teknikler arasında renk bozulması (color jittering), histogram eşitleme ve görüntü normalizasyonu yer almakta olup, modelin çevresel değişkenliklere karşı dayanıklı hale gelmesini sağlamıştır. Eğitim süreci sonucunda model %98,07 doğruluk, %98,15 özgüllük (specificity) ve %99,89 kesinlik (precision) oranlarına ulaşmıştır.Donanım olarak sistem, NVIDIA Jetson AGX Orin üzerinde çalışmakta ve görüntü alımı için BASLER acA2500-60uc kamera kullanılmaktadır. Görüntüleme ve analiz süreci tamamlandığında sınıflandırma sonucu, ModBus protokolü ile bir PLC'ye iletilmekte ve sistem otomatik karar alabilmektedir. Ayrıca, Raspberry Pi tabanlı bir ekran ile sistemin durumu kullanıcıya görsel olarak da aktarılmaktadır. Uygulama, çapaklı yüzeyler, ışık yansımaları ve titreşim gibi üretim ortamındaki zorlu koşullar altında test edilmiştir.Sonuç olarak, bu çalışma yalnızca akademik anlamda değil, pratik endüstriyel uygulama açısından da yüksek değer taşımaktadır. Yapay zekânın üretim sahasında doğrudan kullanımı ile hem insan kaynaklı hatalar azaltılmış hem de kalite kontrol süreci otomatikleştirilmiştir. Bu bağlamda çalışma, literatürde az sayıda bulunan gerçek saha uygulamaları arasında yer almakta ve Endüstri 4.0–5.0 vizyonuna somut katkı sağlamaktadır.

ARTIFICAL INTELLIGENCE AND IMAGE PROCESSING-BASED PART FEEDING CONTROL IN A ROBOT CELL

SUMMARY

In recent years, extensive literature has highlighted the challenges associated with conventional visual inspection systems in industrial settings. Traditional machine vision approaches—such as template matching, Hough transforms, and contour-based techniques—tend to perform adequately under strictly controlled laboratory conditions but often fail when deployed in dynamic environments. Numerous studies in academic literature emphasize the limitations of fixed rule-based systems, particularly under fluctuating lighting, varying object orientations, or the presence of noise and occlusion. These methods typically rely on handcrafted features and are prone to generalization failures. While artificial neural networks (ANNs) and traditional support vector machines (SVMs) have been introduced as alternatives, they too suffer from scalability issues and the need for manual feature extraction.

More recent research has turned to convolutional neural networks (CNNs) and deep learning-based object detection architectures. Models such as Faster R-CNN, SSD, and YOLO have been benchmarked across diverse datasets like COCO and PASCAL VOC, demonstrating superior adaptability to real-world variability. However, most of these implementations have been confined to academic or synthetic datasets, with limited validation under real industrial constraints. This research seeks to bridge that gap by delivering a fully implemented, real-time system validated within an operational manufacturing environment.

This research addresses a pressing challenge in modern manufacturing—part feeding errors in automated production lines. These errors, particularly in high-precision sectors such as automotive manufacturing, can result in defective assemblies, production halts, and increased costs. To mitigate this issue, an artificial intelligence (AI)-powered real-time visual inspection system was developed and implemented in a real industrial setting. The core of this system is the YOLOv7-tiny model, a fast and lightweight convolutional neural network optimized for edge device deployment.

Traditional machine vision systems rely heavily on handcrafted features and fixed rule-based algorithms such as thresholding, edge detection, and contour analysis. While useful in controlled environments, these methods fail under varying illumination, part orientation, dust, vibrations, and background complexity. In contrast, deep learning-based solutions offer better generalization and adaptability by learning hierarchical features directly from data. However, such solutions are rarely applied and validated under real manufacturing conditions due to computational and data constraints. This study fills that gap.

YOLOv7-tiny was selected due to its balance of computational efficiency and high detection accuracy. Alternative models, such as SSD and Faster R-CNN, were evaluated; SSD lacked robustness in detecting small or partially occluded objects, while Faster R-CNN demanded higher GPU resources unsuitable for embedded

platforms. YOLOv7-tiny's architecture—comprising a CSPDarknet backbone, PANet neck, and decoupled head—enables real-time object detection with minimal latency.

A total of 2400 labeled images were collected directly from a functioning production cell. These images included an equal distribution of OK and NG samples. Since NG parts naturally occur less frequently, a series of advanced data augmentation techniques was employed, including color jittering, Gaussian noise, histogram equalization, rotation, scaling, and synthetic NG generation using compositing methods. This not only balanced the dataset but also enhanced robustness against environmental variability.

The model was trained using the Darknet framework on a CUDA-enabled workstation. All dependencies, including cuDNN, OpenCV, and NumPy, were installed. GPU acceleration was activated and training parameters—batch size, learning rate, momentum—were carefully tuned. The loss function integrated objectness confidence, bounding box regression, and class prediction. Training was monitored using validation loss and early stopping techniques to prevent overfitting. The final model achieved 98.07% accuracy, 99.89% precision, 98.08% recall, and 98.15% specificity on the test dataset.

The trained model was deployed on an NVIDIA Jetson AGX Orin, capable of real-time processing with low energy consumption. An industrial-grade BASLER acA2500-60uc camera was installed to capture images at precise intervals. System integration was achieved via ModBus TCP/IP using an ADAM-6052 module to establish communication between the AI unit and the PLC. A touchscreen interface based on WaveShare technology allowed operators to monitor system output and receive alerts.

During production, the PLC triggers the system upon receiving a "go to robot" signal. The AI module captures an image, analyzes part placement, and determines OK or NG status. In the case of an NG detection, the system sends an immediate signal to halt robotic action or notify the operator, thus preventing defective parts from advancing further into the process chain. The entire detection and response cycle completes in less than 200 ms.

The system was validated in a challenging industrial environment subject to noise, dust, and lighting fluctuations. It performed robustly across all conditions. Performance was measured using confusion matrices, and in several validation rounds, NG detection achieved 100% sensitivity—crucial for critical failure prevention.

Additional preprocessing modules were incorporated to optimize input data quality and enhance detection accuracy. Grayscale conversion reduced color variation and computational complexity. Noise filtering helped to eliminate camera artifacts and background disturbances. Histogram equalization improved contrast in low-light scenarios, making the features more distinguishable. Normalization scaled pixel values into a uniform range, ensuring stable learning. These stages collectively improved the model's consistency and robustness.

The model's high performance is quantitatively demonstrated by its evaluation metrics:

Accuracy (98.07%): The overall correctness of predictions across all classes.Precision (99.89%): The proportion of parts predicted as OK that were actually OK, minimizing false positives.Recall (98.08%): The percentage of actual OK parts that were correctly identified by the model.Specificity (98.15%): The ability of the model to correctly identify NG parts and minimize false negatives.

Each of these metrics contributes uniquely to evaluating the system. In a manufacturing setting, high specificity is critical to ensuring that defective parts (NG) are not overlooked. High precision prevents unnecessary halts for parts that are actually OK. The balance of all metrics demonstrates the system's suitability for real-world deployment.

The academic contribution of this study lies in demonstrating that AI-based visual inspection can move beyond the lab and be applied in complex, dynamic manufacturing environments. Unlike prior studies restricted to synthetic or laboratory data, this system was trained, tested, and deployed in the field. It demonstrates how deep learning can be integrated with hardware controllers and existing industrial protocols to create a seamless, intelligent automation solution.

This research also introduces a novel strategy for compensating data scarcity—synthetic image generation for underrepresented defect classes. This approach significantly improved model generalization, contributing to consistent performance despite data imbalance. The integration with PLC systems and usage of real-time monitoring makes the solution immediately deployable in Industry 4.0 and 5.0 contexts.

In conclusion, this work establishes a full-cycle AI solution from data acquisition to field deployment. It validates YOLOv7-tiny’s effectiveness in low-latency, high-accuracy scenarios and underscores the practical feasibility of deploying AI at the edge in industrial environments. The methodology, tools, and results form a valuable reference for future implementations of intelligent visual inspection systems.

1. GİRİŞ

1.1. Çalışmanın Amacı

Bu çalışmanın amacı, endüstriyel üretim hatlarında parça besleme sürecinde oluşabilecek hataları minimize etmek ve kalite kontrolünü iyileştirmek için yapay zeka destekli bir görüntü işleme sistemi geliştirmektir. Geleneksel görüntü işleme yöntemlerinin yetersizlikleri göz önünde bulundurularak, derin öğrenme tabanlı bir model olan YOLOv7-tiny kullanılarak gerçek zamanlı nesne tespiti sağlanmıştır. Özellikle otomotiv sektöründe, üretim süreçlerinde insan kaynaklı hatalar nedeniyle yanlış parça besleme ve montaj hataları meydana gelmektedir. Bu hatalar, üretim maliyetlerini artırmakta ve kalite standartlarını olumsuz yönde etkilemektedir. Bu çalışmada, otomasyon sistemleri ile entegre çalışabilen bir yapay zeka modeli kullanılarak, parçaların doğru şekilde yerleştirilip yerleştirilmediğini tespit etmek hedeflenmiştir. Bu doğrultuda, geliştirilen sistemde ModBus protokolü ile PLC haberleşmesi sağlanarak üretim hattıyla entegrasyon gerçekleştirilmiş, NVIDIA Jetson AGX Orin donanımı üzerinde çalışan model ile yüksek doğrulukta ve hızlı bir şekilde parça kontrolü yapılması amaçlanmıştır. Ayrıca, sistemin çevresel faktörlerden etkilenmemesi için veri artırma teknikleri kullanılarak modelin güvenilirliği artırılmıştır. Sonuç olarak, bu çalışma endüstriyel otomasyon süreçlerinde yapay zeka ve görüntü işleme teknolojilerinin etkinliğini artırarak üretim kalitesini yükseltmeyi, insan hatalarını en aza indirmeyi ve verimliliği artırmayı hedeflemektedir.

1.2. Problemin Tanımı

Endüstriyel üretim süreçlerinde parça besleme hataları, kalite ve verimlilik açısından önemli sorunlara yol açmaktadır. Özellikle otomotiv sektöründe, robotlar ve operatörler tarafından beslenen parçaların yanlış yerleştirilmesi üretim sürecinde kalite düşüşüne, hurda oranlarının artmasına ve maliyetlerin yükselmesine neden olmaktadır. Geleneksel üretim hatlarında insan gözetimine dayalı kalite kontrol süreçleri kullanıldığından, insan kaynaklı hatalar kaçınılmaz hale gelmektedir. Mevcut sistemlerde, yanlış beslenen parçalar genellikle bir sonraki montaj aşamasında fark edilmekte ve bu durum hem zaman kaybına hem de maliyet artışına sebep olmaktadır.

Ayrıca, operatörlerin sürekli olarak parçaları manuel olarak kontrol etmesi, üretim hızını düşürmekte ve iş gücü gereksinimini artırmaktadır. Robot destekli üretim hatlarında bile yanlış parça besleme sorunu yaşanabilmekte, bu da üretim sürecinin güvenilirliğini olumsuz etkilemektedir. Geleneksel görüntü işleme yöntemleri, parçaların doğru yerleştirilip yerleştirilmediğini tespit etmekte yetersiz kalmaktadır. Işık değişimleri, farklı açılardan gelen gölgeler ve üretim ortamındaki zorlu koşullar (örneğin kaynak sırasında oluşan çapaklar ve duman) bu yöntemlerin güvenilirliğini azaltmaktadır. Bu nedenle, gerçek zamanlı ve yüksek doğruluk oranına sahip bir görüntü işleme sistemi ihtiyacı doğmuştur. Bu çalışmada, YOLOv7-tiny derin öğrenme modeli kullanılarak, parça besleme sürecinde yanlış yerleştirilen parçaların tespiti amaçlanmıştır. Sistem, ModBus protokolü ile PLC haberleşmesi sağlayarak üretim hattıyla entegre olacak ve operatörlere hataları önceden bildirecek şekilde tasarlanmıştır. Bu sayede, insan hatalarının minimize edilmesi, üretim sürecinin hızlandırılması ve kalite kontrol süreçlerinin iyileştirilmesi hedeflenmektedir.

2. LİTERATÜR TARAMASI

2.1. Endüstriyel Görüntü İşleme Sistemleri

Endüstriyel üretim süreçlerinde kalite kontrol, hata tespiti ve süreç optimizasyonu önemli bir rol oynamaktadır. Endüstri 4.0 ve 5.0 ile birlikte üretim hatlarının dijitalleşmesi, yapay zeka ve görüntü işleme tekniklerinin entegrasyonunu zorunlu hale getirmiştir [1]. Endüstriyel görüntü işleme sistemleri, üretim hatlarında insan müdahalesine gerek kalmadan otomatik nesne tanıma ve hata tespiti yaparak süreçleri optimize etmektedir [2]. Bu tür sistemler, seri üretimde kalite kontrol süreçlerini hızlandırarak hata oranlarını düşürmekte ve üretim verimliliğini artırmaktadır. Xu, Ragot ve Dupuis ,endüstriyel nesne tanıma sistemlerinde görüş açısının doğruluk üzerindeki etkisini inceleyerek, doğru kamera konumlandırmasının kalite denetimini nasıl iyileştirdiğini ortaya koymuştur [1]. Shrestha, Karki, Yonjan, Subedi ve Phuyal ise otomatik nesne tespiti ve ayrıştırma sistemlerinin üretim süreçlerini hızlandırabileceğini göstermiştir [2]. Endüstride görüntü işleme sistemlerinin en yaygın kullanım alanlarından biri otomatik kalite kontrol sistemleridir. Boukouvalas ve diğerleri seramik karo üretiminde yüzey kusurlarını tespit eden bir sistem geliştirmiştir [3]. Bu sistem, karo yüzeyindeki çatlakları, lekeleri ve diğer kusurları otomatik olarak algılayarak kalite denetimini iyileştirmiştir. Benzer şekilde, Abbas, Shakoor, Khan, Ahmed ve Khurshid ,tarım ürünlerinin sınıflandırılması ve kalite kontrolü için görüntü işleme tabanlı bir sistem önermiştir [4]. Ancak, bu tür basit görüntü işleme yöntemleri, değişken çevresel faktörler nedeniyle yetersiz kalmaktadır.

2.1.1. Endüstriyel görüntü işleme yöntemlerinin yetersizlikleri

Geleneksel görüntü işleme yöntemleri, endüstriyel üretim süreçlerindeki karmaşık senaryolar karşısında yetersiz kalabilmektedir. Özellikle düşük ışık koşulları, nesneler arasındaki ince farklılıklar ve üretim hatlarındaki çevresel faktörler, sistemlerin doğruluk oranlarını olumsuz etkilemektedir [5]. Bu eksiklikler dört temel başlık altında toplanabilir.

2.1.1.1. Parça tanıma ve sınıflandırma problemleri

Geleneksel görüntü işleme yöntemleri, üretim hattında yer alan parçalar arasındaki küçük farkları ayırt etmekte yetersiz kalabilmektedir [5]. Bu durum, özellikle otomotiv sektöründe kullanılan metal parçalar için daha belirgindir. Çünkü bu parçalar genellikle birbirine çok benzer yüzey özelliklerine sahiptir. Bu benzerlik, standart görüntü işleme tekniklerinin parçalar arasındaki farklılıkları algılamasını zorlaştırmakta ve sınıflandırma doğruluğunu düşürebilmektedir [6].

2.1.1.2. Kusur algılama ve segmentasyon problemleri

Basit görüntü işleme teknikleri, üretim hattında meydana gelen küçük çatlaklar, yüzey bozulmaları ve kusurlar gibi ince detayları tespit etmede yeterli hassasiyeti sağlayamamaktadır [7]. Ayrıca, yanlış beslenen ya da eksik monte edilen parçaların zamanında ve doğru şekilde belirlenmesi, geleneksel yöntemler açısından önemli bir zorluk teşkil etmektedir [8] .

2.1.1.3. Gürültü ve çevresel faktörlere duyarlılık

Üretim ortamlarında sıkça karşılaşılan toz, çapak, duman ve değişken aydınlatma koşulları, geleneksel görüntü işleme sistemlerinin doğruluk ve kararlılık seviyelerini ciddi şekilde düşürmektedir [9]. Özellikle kaynak işlemlerinin yoğun olduğu fabrikalarda görülen yüksek sıcaklık ve parlama gibi etkiler, optik sensörlerin hassasiyetini azaltarak sistemin güvenilirliğini olumsuz yönde etkileyebilmektedir [10].

2.1.1.4. Gerçek zamanlı işlem gereksinimi

Endüstriyel üretim hatlarında kararların anlık olarak verilmesi büyük önem taşımaktadır. Ancak geleneksel görüntü işleme yöntemleri bu ihtiyaca zaman açısından yeterince karşılık verememektedir [11]. Özellikle yüksek hızlı üretim süreçlerinde, milisaniyeler içinde doğru kararlar alabilen sistemlerin kullanımı kaçınılmaz hale gelmiştir [12]. Bu sorunların üstesinden gelebilmek için görüntü işleme sistemlerinin derin öğrenme tabanlı çözümlerle desteklenmesi gerekmektedir. Geleneksel sistemler, sınırlı algoritma yapıları nedeniyle üretim hatlarında güvenilir bir performans sergileyememektedir. Bu nedenle, gelişmiş nesne tanıma algoritmaları ve yapay zeka destekli görüntü işleme çözümleri, endüstriyel otomasyon süreçlerinde giderek daha fazla tercih edilmektedir [13].

2.1.2. Endüstriyel görüntü işleme sistemlerinin otomasyon ile entegrasyonu

Endüstriyel görüntü işleme sistemlerinin etkili bir şekilde çalışabilmesi için, otomasyon sistemleri ile entegre edilmesi gerekmektedir. PLC (Programmable Logic Controller) tabanlı kontrol sistemleri ile entegre çalışan görüntü işleme çözümleri, üretim süreçlerini daha verimli hale getirmektedir. ModBus ve Ethernet TCP/IP gibi haberleşme protokolleri, üretim hattındaki makinelerle görüntü işleme sistemleri arasında veri akışını sağlayarak gerçek zamanlı kalite kontrol mekanizmalarının oluşturulmasına olanak tanımaktadır [14]. Görüntü işleme sistemlerinin otomasyon hatlarıyla entegrasyonu, hata tespit süreçlerini hızlandırarak insan müdahalesine duyulan ihtiyacı azaltmaktadır. Geleneksel olarak, kalite kontrol işlemleri operatörler tarafından manuel olarak gerçekleştirildiğinde zaman kaybına ve yüksek hata oranlarına neden olmaktadır. Ancak, görüntü işleme sistemlerinin PLC ile entegre çalışması sayesinde üretim hatları otomatik olarak hata tespiti yapabilmekte ve sistemin hata durumlarına anında tepki vermesi sağlanmaktadır [15].

2.2. Nesne Tespit Algoritmaları ve YOLOv7-Tiny Modeli

2.2.1. Geleneksel nesne tespit yöntemleri

Endüstriyel nesne tespitinde kullanılan ilk yöntemler geleneksel görüntü işleme tekniklerine dayanmaktadır. Bu yöntemlerde, nesne tespiti için kenar belirleme, renk eşleme, kontur analizi gibi manuel özellik çıkarma teknikleri kullanılmaktadır. Binyan, Yanbo, Zhihong, Jiayu ve Junqin [15], endüstriyel üretim ortamlarında nesne tespiti ve robotik sıralama sistemlerini inceledikleri çalışmalarında, manuel görüntü işleme prosedürlerinin çeşitli çevresel faktörlerden etkilendiğini ve hata oranlarını artırdığını göstermiştir. Geleneksel yöntemler içerisinde, özellikle SVM (Support Vector Machine) ve Template Matching gibi teknikler, kalite kontrol süreçlerinde yaygın olarak kullanılmıştır. Ancak, bu yöntemler büyük ölçekli endüstriyel uygulamalar için yeterli doğruluk ve hız sağlayamamaktadır. Özellikle aydınlatma değişimleri, gürültü seviyeleri ve üretim sürecinde meydana gelen değişkenler, geleneksel görüntü işleme yöntemlerinin doğruluğunu düşürmektedir [15]. Bu problemlerin aşılması için Artificial Neural Networks (ANN) gibi öğrenme tabanlı yaklaşımlar geliştirilmiştir. ANN tabanlı sistemler, görüntüden elle özellik çıkarma gereksinimini ortadan kaldırarak modelin otomatik olarak öğrenmesini sağlamaktadır. Ancak, ANN sistemlerinin karmaşık entegrasyon süreçleri ve yüksek veri ihtiyacı, üretim sahalarında uygulanmasını zorlaştırmaktadır [16].

2.2.2. Derin Öğrenme ve Nesne Tespit Algoritmalarının Evrimi

Derin öğrenmenin gelişimi ile birlikte, geleneksel özellik çıkarma yöntemlerinin yerine tamamen veri odaklı nesne tespit sistemleri geliştirilmiştir. 2012 yılında AlexNet modeli ile başlayan derin öğrenme devrimi, 14 milyon görsel ve 21,841 kategori üzerinde eğitilerek nesne tanımada yüksek başarı sağlamıştır [16]. Bu gelişme ile birlikte, araştırmacılar RCNN, Fast-RCNN, Faster-RCNN, SSD ve YOLO gibi nesne tespit algoritmalarını geliştirmiştir. Özellikle R-CNN (Region-based Convolutional Neural Network) tabanlı sistemler, nesne tespiti konusunda önemli doğruluk oranlarına ulaşmış, ancak bu yöntemlerin işlem süresi çok uzun olduğu için gerçek zamanlı uygulamalarda etkili bir çözüm sunamamıştır. Zuo, Wang ve Song kaynak yüzeylerindeki kusurların tespiti için YOLO tabanlı sistemler ile Faster R-CNN modelini karşılaştırarak, YOLO algoritmasının hem hız hem de doğruluk açısından üstün performans gösterdiğini raporlamıştır [16]. Bu tür araştırmalar sonucunda, özellikle hızın kritik olduğu endüstriyel süreçlerde, tek aşamalı (one-stage) nesne tespit algoritmalarına olan ilgi artmıştır. Bu bağlamda, YOLO (You Only Look Once) algoritması, tek bir işlemde nesne tespiti yapabilmesi nedeniyle yüksek hız ve doğruluk sağlamaktadır.

2.2.3. YOLO Serisinin evrimi ve yolov7-tiny modeli

YOLO algoritması ilk olarak Joseph Redmon ve Ali Farhadi tarafından geliştirilmiş ve 2016 yılında tanıtılmıştır [5]. Algoritma, çok aşamalı nesne tespit yöntemlerine kıyasla doğrudan tüm görüntüyü analiz ederek nesne tespitini hızlandırmıştır.

YOLO'nun gelişimi şu aşamalardan geçmiştir:

- **YOLOv1** (2016): İlk sürüm olup, hızlı ancak doğruluk açısından eksiklikleri vardır [5].
- **YOLOv2 ve YOLO9000** (2017): Daha küçük nesneleri tespit edebilmek için iyileştirilmiş modeldir [8].
- **YOLOv3** (2018): Darknet-53 derinlikli bir mimari kullanarak doğruluğu artırmış ancak hız açısından yeterli gelişimi sağlayamamıştır [16].
- **YOLOv4 ve YOLOv5** (2020): Ölçeklenebilirlik ve hız açısından geliştirilmiş versiyonlar olup, ticari kullanım alanlarında yaygın olarak tercih edilmiştir.

- **YOLOv6 ve YOLOv7** (2022): Modern GPU mimarileriyle optimize edilen ve günümüz endüstriyel uygulamalarında en yüksek doğruluk oranlarına sahip olan modellerdir.

YOLOv7, Faster R-CNN gibi geleneksel yöntemlerden daha iyi doğruluk oranlarına ulaşırken, YOLOv5'ten de daha hızlı çalışmaktadır. Bu bağlamda, endüstriyel üretim hatlarında gerçek zamanlı kalite kontrol ve parça yerleştirme işlemlerinde en uygun modellerden biri olarak öne çıkmaktadır. Özellikle YOLOv7-Tiny modeli, sınırlı işlem gücüne sahip gömülü sistemlerde çalışmak için optimize edilmiştir. Zuo, Wang ve Song [16], YOLOv7-Tiny modelinin hafif yapısı ve yüksek işlem kapasitesi sayesinde endüstriyel robot hücrelerinde kullanılabilecek en uygun nesne tespit modellerinden biri olduğunu ortaya koymuştur.

2.2.4. Yolov7-Tiny modelinin endüstriyel uygulamalardaki avantajları

YOLOv7-Tiny modeli, hafif, hızlı ve yüksek doğruluk oranına sahip bir nesne tespit algoritması olarak, endüstriyel kalite kontrol süreçlerinde avantajlar sunmaktadır. Bu modelin endüstriyel kullanımı aşağıdaki avantajları içermektedir:

2.2.4.1. Düşük hesaplama maliyeti

YOLOv7-Tiny modeli, NVIDIA Jetson AGX Orin, Raspberry Pi gibi gömülü sistemlerde sorunsuz şekilde çalışabilmektedir. Bu sayede, endüstriyel üretim hatlarında yüksek donanım gereksinimi olan sunuculara ihtiyaç duymadan işlem yapılabilmesi mümkün olmakta ve bu durum işletmelere önemli bir maliyet avantajı sunmaktadır.

2.2.4.2. Bir gerçek zamanlı işleme yeteneği:

Model, oldukça düşük gecikme süresiyle çalışarak milisaniyeler içinde karar verebilme yeteneğine sahiptir. Bu sayede, kaynak fabrikaları, montaj hatları ve parça besleme sistemleri gibi hızlı üretim gerektiren ortamlarda anlık kalite kontrolü yapılabilmekte ve olası hatalar üretim süreci sırasında tespit edilerek müdahale edilebilmektedir.

2.2.4.3. Yüksek doğruluk ve güvenilirlik:

Zuo, Wang ve Song tarafından gerçekleştirilen çalışmada, YOLO algoritmasının kaynak yüzeyindeki kusurları tespit etmede yüksek doğruluk oranı ile başarılı sonuçlar verdiği ve %97.2 doğruluk oranına ulaştığı rapor edilmiştir [16]. Bununla birlikte,

modelin yanlış parça yerleştirme, eksik montaj veya hatalı bileşen tespiti gibi endüstriyel üretim süreçlerinde de etkin bir şekilde uygulanabildiği ortaya konmuştur. Bu durum, modelin farklı hata türlerine karşı esnek ve güvenilir bir şekilde çalıştığını göstermektedir.

2.2.4.4. Kolay entegrasyon

Model, ModBus protokolü aracılığıyla PLC (Programmable Logic Controller) sistemlerine entegre edilerek doğrudan üretim hatlarında kullanılabilmektedir. Bu sayede endüstriyel haberleşme sistemleri ile uyumlu bir şekilde çalışarak, kalite kontrol sonuçlarını operatörlere anlık olarak iletebilmekte ve üretim sürecine gerçek zamanlı geri bildirim sağlamaktadır. Bu entegrasyon, hem süreç takibini kolaylaştırmakta hem de hata tespiti sonrası hızlı müdahaleye olanak tanımaktadır. Nesne tespit algoritmalarının gelişimi incelendiğinde, geleneksel yöntemlerin yerini derin öğrenme tabanlı modellerin aldığı ve YOLO algoritmalarının hız açısından büyük avantaj sağladığı görülmektedir. YOLOv7-Tiny modeli, hafif yapısı, yüksek işlem kapasitesi ve doğruluğu sayesinde endüstriyel üretim süreçlerinde kullanılabilecek en uygun modellerden biri olarak öne çıkmaktadır. Bu çalışmada geliştirilen sistemde, YOLOv7-Tiny modeli kullanılarak parça besleme işlemlerinde kalite kontrol mekanizması oluşturulmuş ve endüstriyel otomasyon sistemlerine entegre edilmiştir. Modelin yüksek hızda, düşük maliyetle ve yüksek doğrulukla çalışması, üretim hatlarında insan hatalarının en aza indirilmesine ve süreç verimliliğinin artırılmasına olanak sağlamaktadır.

3. MATERYAL VE YÖNTEM

3.1. Sistem Genel Mimarisi

Bu çalışmada geliştirilen sistem, üretim hattındaki parça besleme sürecini izlemek ve kontrol etmek amacıyla görüntü işleme, yapay zeka ve otomasyon teknolojilerinin entegre bir yapıda kullanılmasıyla oluşturulmuştur. Sistem; endüstriyel kamera ve lens ile görüntü alımı, Jetson AGX Orin platformu üzerinde çalışan derin öğrenme modeli ile analiz, PLC üzerinden ModBus protokolü ile haberleşme ve dokunmatik operatör arayüzü ile kullanıcı etkileşimi sağlayan dört temel bileşenden oluşmaktadır.

Kamera ve lens kombinasyonu, yüksek çözünürlük ve düşük ışık koşullarına uygunluk açısından tercih edilmiştir. Görüntülerin analizi, gerçek zamanlı çalışabilen yapay zeka işlemcisi ile gerçekleştirilmiştir. Sistem, üretim hattı üzerindeki PLC ile dijital veri alışverişini ADAM-6052 modülü aracılığıyla sağlamaktadır. Kullanıcıların süreci izleyebilmesi ve gerektiğinde müdahale edebilmesi için dokunmatik ekran arayüzü entegre edilmiştir.

Bu dört ana bileşen, birbirleriyle senkronize çalışacak şekilde yapılandırılmıştır ve üretim sürecinin doğruluk, hız ve güvenilirlik açısından iyileştirilmesini hedeflemektedir. Donanımın seçim gerekçeleri, yazılım altyapısı ve entegrasyon detayları sonraki bölümlerde sunulacaktır.

3.2. Donanım ve Yazılım Altyapısı

Bu bölümde sistemde kullanılan donanım bileşenleri ile yazılım altyapısının genel yapısı açıklanmakta, yapay zeka modelinin eğitimi ve entegrasyon süreci özetlenmektedir.

3.2.1. Donanım bileşenleri

Sistemin geliştirilmesinde, sahada güvenilir çalışabilecek, gerçek zamanlı analizleri destekleyecek donanım bileşenleri tercih edilmiştir. Seçilen donanımlar; görüntü kalitesi, işlem kapasitesi, haberleşme uyumluluğu ve kullanıcı arayüzü gibi kriterler

göz önünde bulundurularak belirlenmiştir. Tablo 3.1`de sistemde kullanılan temel donanımlar ve görevleri özetlenmiştir:

Tablo 3.1. Sistemde Kullanılan Donanım Bileşenleri ve Görevleri

Bileşen	Model / Özellik	Görev
Endüstriyel Kamera	Basler acA2500-60uc	Yüksek çözünürlükte görüntü alma
Lens ve Filtre	LM50-HC +50.6mm UV filtre	Netlik ve optik koruma
Yapay Zeka İşlemci	NVIDIA Jetson AGX Orin	Derin öğrenme modeli çalıştırma
Haberleşme Modülü	Advantech ADAM-6052	ModBus TCP üzerinden veri iletimi
Operatör Arayüzü	10.1" dokunmatik ekran	Kullanıcı izleme ve etkileşim

Bu donanımlar birbiriyle senkronize şekilde çalışacak biçimde yapılandırılmıştır. Jetson AGX Orin, YOLOv7-tiny modelini düşük gecikme süresiyle çalıştırabilecek kapasiteye sahip olup, gerçek zamanlı analiz ihtiyacını karşılamaktadır. Endüstriyel kamera ve lens kombinasyonu ile farklı ışık koşullarında bile yüksek kaliteli görüntüler alınabilmiş, veri seti oluşturma ve analiz süreçlerinin güvenilirliği artırılmıştır. Haberleşme modülü üzerinden alınan dijital tetikleme sinyalleri ile süreç otomatik olarak başlatılmakta ve karar çıktısı yine aynı modül üzerinden PLC'ye iletilmektedir. Operatör ekranı aracılığıyla sistemin durumu takip edilebilmekte ve gerektiğinde manuel müdahale yapılabilmektedir.

3.2.2. Yazılım altyapısı ve yapay zeka modeli

Sistemin yazılım altyapısı Python programlama dili ile geliştirilmiştir. Görüntü işleme, yapay zeka modeli entegrasyonu, kullanıcı arayüzü ve PLC haberleşmesi olmak üzere tüm yazılım modülleri bağımsız bileşenler halinde yapılandırılmış ve ana kontrol yazılımı altında birleştirilmiştir.

YOLOv7-tiny modeli, görüntülerdeki parçaların doğru yerleştirilip yerleştirilmediğini tespit etmek amacıyla kullanılmıştır. Modelin eğitimi Python tabanlı PyTorch framework'ü ile gerçekleştirilmiş, eğitim sürecinde CUDA desteği ile GPU hızlandırma sağlanmıştır. Eğitim aşamasında 2400 adet görüntü kullanılmış; bu görüntüler çeşitli veri artırma (augmentation) teknikleri ile çeşitlendirilerek modelin genelleme kapasitesi artırılmıştır.

Modelin çalışması için gerekli olan görüntü toplama, ön işleme, model çıktısını değerlendirme ve karar üretme gibi süreçler ayrı yazılım modülleri halinde

geliştirilmiş ve NVIDIA Jetson AGX Orin üzerinde çalışacak şekilde optimize edilmiştir. ModBus TCP protokolü üzerinden dijital giriş ve çıkışlar ile haberleşme sağlanmış, operatör arayüzü ile sistem durumu ve model çıktıları görselleştirilmiştir.

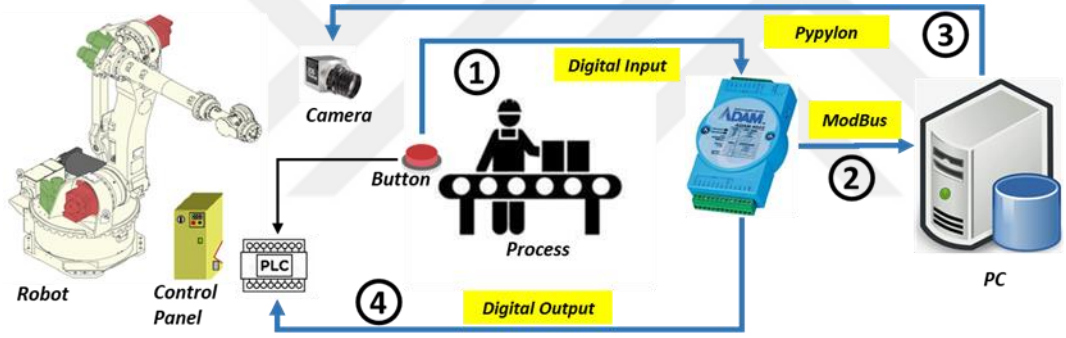
3.3. Donanım Yapılandırılması

Sistemin doğru ve verimli bir şekilde çalışabilmesi için donanım bileşenlerinin birbirleriyle senkronize olarak çalışması gerekmektedir. Bu yapılandırmada, parça tespiti, veri işleme ve karar mekanizmasının üretim hattına entegre edilmesi esas alınmıştır. Donanım süreçleri, sinyal iletimi, görüntü işleme ve karar aktarımı aşamalarından oluşmaktadır. Sistem, operatör tarafından parçanın yerleştirilmesiyle çalışmaya başlamaktadır. Parça konulduğunda, sistemin tetiklenmesi için bir sinyal üretilmekte ve bu sinyal, yazılımın çalıştığı bilgisayara gönderilmektedir. Bilgisayar, bu sinyali aldıktan sonra, kamera sistemine görüntü al komutu göndermekte ve endüstriyel kamera belirlenen çerçevede görüntü yakalamaktadır. Alınan görüntü, Jetson AGX Orin üzerinde çalıştırılan nesne tespit algoritmaları tarafından işlenerek parçanın doğru yerleştirilip yerleştirilmediği tespit edilmektedir. Görüntü işleme süreci tamamlandıktan sonra, elde edilen sonuç üretim hattındaki robot sistemi ile paylaşılmaktadır. Bu sayede, robot sistemi parçanın doğru olup olmadığına göre bir sonraki işlemi gerçekleştirmekte ya da hata durumunda operatöre uyarı vermektedir. Sistemin son aşamasında, işlenen veriler ve tespit sonuçları, Waveshare 10.1-inch Capacitive Touch Display üzerinden görselleştirilerek operatör tarafından takip edilebilmektedir. Operatör, ekran üzerinden süreci izleyerek hata durumlarında müdahalede bulunabilmekte ve sistemin çalışma durumunu değerlendirebilmektedir.

3.3.1. Temel donanım yapılandırılması

Sistemin verimli çalışabilmesi için donanım bileşenlerinin birbirleriyle uyumlu bir şekilde haberleşmesi gerekmektedir. Bu yapılandırma, parça tespiti, veri iletimi, görüntü işleme ve karar mekanizmasının üretim sürecine entegrasyonunu kapsamaktadır. Süreç dört temel adımdan oluşmaktadır. İlk olarak, operatör tarafından parçanın üretim istasyonuna yerleştirilmesiyle birlikte bir tetikleme sinyali üretilmektedir. Bu sinyal, parça konuldu bilgisini iletmek amacıyla kullanılan buton tarafından üretilmekte ve ADAM-6052 haberleşme modülüne aktarılmaktadır. ADAM-6052, dijital girişleri algılayarak PLC ve bilgisayar sistemleri ile haberleşmeyi sağlayan bir arayüz görevi görmektedir. İkinci adımda, ADAM-6052 modülü,

tetikleme sinyalini ModBus/TCP protokolü üzerinden bilgisayara iletmektedir. Bu haberleşme, sistemin operatörden bağımsız ve otomatik olarak çalışmasını sağlamakta, süreçler arasındaki gecikmeyi en aza indirmektedir. Bilgisayar, bu sinyali aldığında, parçanın yerleştirildiğini tespit ederek kamera sistemini görüntü yakalama işlemi için tetiklemektedir. Üçüncü aşamada, bilgisayarda çalışan yazılım, ModBus haberleşmesi ile aldığı tetikleme sinyali doğrultusunda kameraya görüntü alma komutunu göndermektedir. Kamera tarafından yakalanan görüntü, NVIDIA Jetson AGX Orin üzerinde çalışan yapay zeka modeli tarafından işlenmektedir. Model, alınan görüntüyü analiz ederek parçanın doğru yerleştirilip yerleştirilmediğini tespit etmekte ve değerlendirme sonucunu oluşturmaktadır. Son aşamada, elde edilen değerlendirme sonucu, ModBus haberleşmesi üzerinden PLC'ye aktarılmaktadır. PLC, bu veriyi değerlendirerek sistemin bir sonraki aşamasına geçmesine veya hata durumunda operatöre uyarı vermesine olanak tanımaktadır. Bu işlemleri özetleyen görsel şekil 3.1'da görülmektedir.

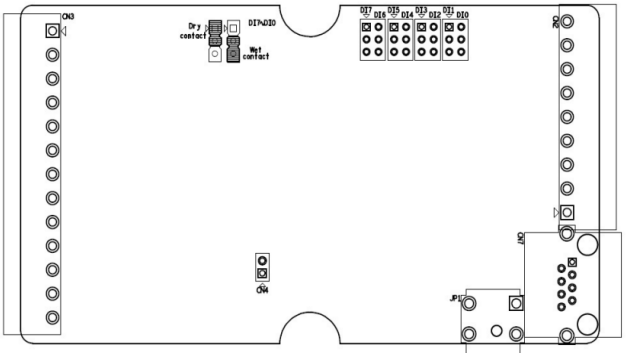


Şekil 3.1. Sistemin hybrid diyagramı

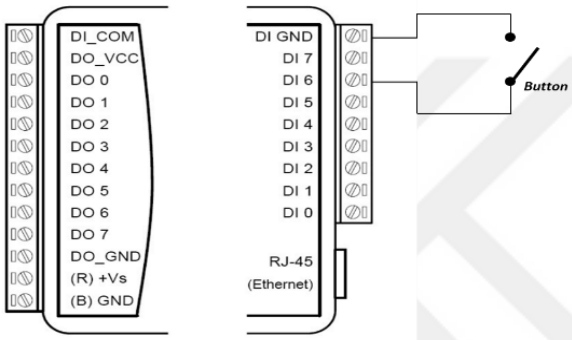
3.3.1.1. Buton tetiğinin PC'ye aktarılması

Sistemin ilk adımında, parçanın üretim istasyonuna yerleştirildiğine dair sinyalin oluşturulması ve iletilmesi gerekmektedir. Bu işlem için butondan gelen tetikleme sinyalinin ADAM-6052 haberleşme modülü aracılığıyla bilgisayara aktarılması sağlanmıştır. Tetikleme sinyalini kuru kontak (dry contact) olarak almak için, öncelikle ADAM-6052 modülünde jumper konfigürasyonu yapılmıştır. Şekil 3.2'de konfigurasyonun nasıl yapıldığı gösterilmiştir. Bu modül, wet (beslemeli) veya dry (kuru kontak) giriş modlarında çalışabilmektedir. Kuru kontak bağlantısı sağlamak için, modül üzerindeki jumperlar dry contact moduna ayarlanmıştır. Konfigürasyon işleminin ardından, buton ile ADAM-6052 arasında kablolama yapılarak bağlantı tamamlanmıştır. Kablolama şekil 3.3'de görülmektedir.

Jumper Setting for Dry and Wet contact:



Şekil 3.2. Adam 6052 jumper konfigurasyonu

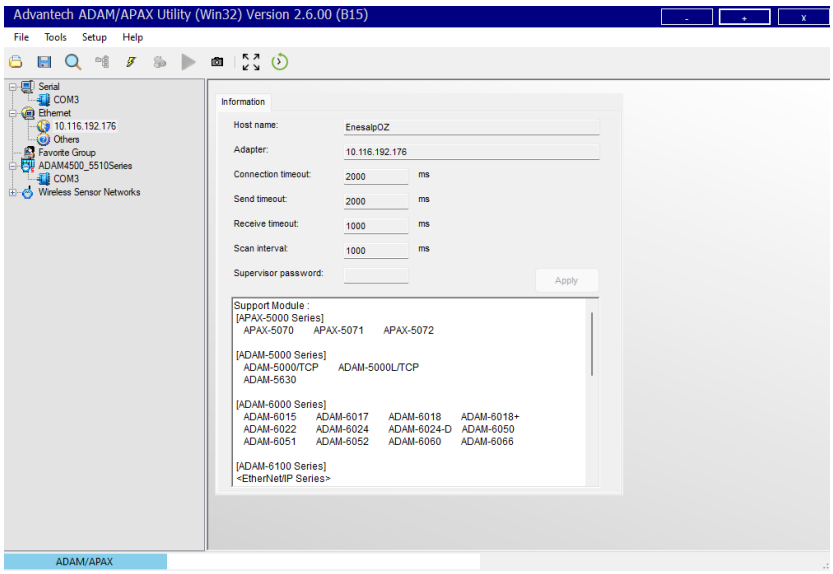


Şekil 3.3. Modül ve buton arası kablolama

Bu bağlantı sayesinde, butona basıldığında ADAM-6052 üzerinden bir dijital giriş sinyali üretilmekte ve sistemin bir sonraki aşamasına geçmesini sağlayan tetikleme işlemi gerçekleştirilmektedir.

3.3.1.2. ADAM-IO 6052 yapılandırma ve veri aktarımı

Bu modülün PC'ye veri aktarımını sağlayabilmek için uygun şekilde yapılandırılması gerekmektedir. Yapılandırma süreci, **ADAM/APAX Utility** programı aracılığıyla gerçekleştirilir ve modülün ağ ayarlarından haberleşme protokolüne kadar çeşitli parametrelerin doğru şekilde tanımlanmasını içerir. ADAM/APAX Utility programının görseli şekil 3.4'te görülmektedir.



Şekil 3.4. ADAM/APAX Utility

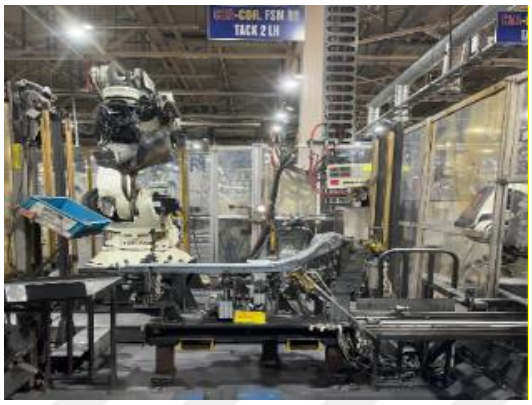
ADAM-IO 6052 modülü, Ethernet veya RS-485 seri bağlantısı üzerinden PC'ye bağlanmalı. Cihazın algılanabilmesi için ADAM/APAX Utility programı kullanılarak bir cihaz taraması gerçekleştirilir. Model otomatik olarak algılanır. Modülün ağ üzerindeki iletişimini sağlamak için IP adresi yapılandırılmalıdır. Varsayılan olarak DHCP özelliği aktif olabilir, ancak endüstriyel uygulamalarda statik IP adresi kullanılması gerekir. ADAM-IO 6052 modülü, dijital giriş (DI) ve dijital çıkış (DO) kanallarına sahiptir. Yapılandırma sırasında, kanalların çalışma modları belirlenir ve giriş/çıkış parametreleri ayarlanır. Örnekleme süresi, giriş sinyallerinin okunma frekansı ve çıkış kontrol modları gibi ayarlar, sistemin ihtiyacına göre düzenlenir. Cihazın PC veya PLC gibi kontrol birimleri ile haberleşebilmesi için uygun bir protokol seçilmelidir. Modbus TCP/IP (Ethernet tabanlı) veya Modbus RTU (RS-485 tabanlı) protokollerinden biri tercih edilerek, haberleşme parametreleri (baud rate, parity, stop bit gibi) endüstriyel ağ yapısına uygun şekilde yapılandırılır. Bu şekilde digital input olarak gelen sinyal ADAM-IO aracılığıyla Modbus TCP/IP protokolüyle de veri aktarımı sağlanır.

3.3.1.3. Basler kamera yapılandırma ve PC ile haberleşme

Kamera, USB 3.0 bağlantısı üzerinden kontrol edildiğinden, IP adresi ayarlamaya gerek yoktur. Ancak, kameranın PC ile uyumlu çalışmasını sağlamak için Basler'in kendi yazılımı olan Pylon Camera Software Suite kullanılarak yapılandırılması gerekmektedir.

3.4. Robotik Üretim Hücresi

Bu bölümde, giriş bölümünde bilgisi verilen FSM RR Tack 2 LH robot prosesi detaylandırılacaktır. Sistem kurulumu, kaynak fabrikasının yan proseslerinden biri olan Front Side Member Rear Tack 2 robot prosesinde gerçekleştirilmiştir. FSM RR'nin görüntüsü şekil 3.5'da görülmektedir.



Şekil 3.5. FSM RR Tack 2 LH

Bu proses, otomotiv üretim sürecinde shell body'yi oluşturan side member parçalarının birleştirilmesini sağlamaktadır. FSM RR Tack 2 prosesinin sağ (RH) ve sol (LH) olmak üzere iki ayrı versiyonu bulunmaktadır ve her iki süreçte de benzer işlemler yürütülmektedir. Proses, iki farklı parçanın birleştirilmesine dayanmaktadır.2 farklı sağa ve sola göre değişen parça ana parçaya birleştirilmektedir. Ana parçanın görüntüsü şekil 3.6'de görülmektedir.

Şekil 3.6. Ana parçanın görüntüsü

Ana parça, her iki tarafta ortak olup, ikinci parça sağ veya sol bölgesine göre farklılık göstermektedir. Operatör tarafından gerçekleştirilen manuel besleme sürecinde, yanlış parça besleme riski bulunmaktadır. Bu durum, yanlış parçanın birleştirilmesine ve

üretim sürecinde hatalı gövde oluşmasına yol açmaktadır. Yanlış besleme sorunu, üretim hattında ancak shell body ile birleştirme aşamasında tespit edilebilmekte, bu da geri dönüşü zor olan hatalara ve hurda oranlarının artmasına neden olmaktadır.

3.5. Veri Toplama Süreci

Basler marka endüstriyel kamera kullanılarak, gerçek zamanlı görüntüler yakalanmış ve daha sonra görüntü işleme teknikleriyle analiz edilmiştir. Veri toplama aşamasında kameradan alınan görüntüler belirli aralıklarla kaydedilmiş ve farklı ışık koşulları, açılar ve ortam değişkenleri dikkate alınarak çeşitlendirilmiştir. Böylece modelin genelleme yeteneği artırılmıştır. Ayrıca, kameranın diyafram ayarı değiştirilerek farklı parlaklıklarda görüntü alınmış, böylece modelin farklı ışık seviyelerinde de doğru çalışabilmesi için veri çeşitliliği artırılmıştır.

Ek A`daki veri toplama kodu, Basler kameradan alınan görüntüleri kaydetmek için Pylon SDK ve OpenCV kütüphanelerini kullanmaktadır. Kodun işleyişi şu şekilde. Kamera başlatılır ve gerçek zamanlı görüntü yakalama işlemi başlar. Yakalanan kareler (frame) belirli aralıklarla kaydedilir. Dosya adı, tarih ve saat bilgisiyle oluşturulur ve ilgili dizine kayıt edilir. Kullanıcı "q" tuşuna bastığında, kamera kapanır ve tüm işlemler durdurulur. Bu süreç, otomatik veri toplama ve saklama işlevini yerine getirerek, yapay zeka modelinin eğitimi için geniş bir veri seti oluşturmayı sağlamıştır.

3.6. Veri Çoğaltma

Veri çoğaltma, derin öğrenme modellerinin eğitim süreçlerinde daha geniş ve çeşitli bir veri seti oluşturmak için uygulanan bir tekniktir. Bu süreç, modelin farklı görüntü koşullarında daha iyi genelleme yapabilmesini sağlamak amacıyla gerçekleştirilir. Yapılan çalışmada, mevcut veri seti üzerinde çeşitli görüntü işleme teknikleri uygulanarak veri artırımı sağlanmıştır. Bu çalışmada kullanılan veri çoğaltma yöntemleri arasında gürültü ekleme, gürültü azaltma, histogram eşitleme, renk dönüşümleri, renk bozulmaları ve kenar algılama işlemleri bulunmaktadır. Uygulanan yöntemlerle modelin farklı ışık koşulları, kontrast seviyeleri ve gürültülü ortamlarda daha başarılı tahminler yapması hedeflenmiştir. Gürültü ekleme işlemi kapsamında Gaussian ve Tuz & Biber gürültüsü uygulanarak modelin rastgele değişen görsel koşullara karşı daha dayanıklı hale getirilmesi sağlanmıştır. Gürültü azaltma teknikleri ile modelin düşük kaliteli görüntülerde daha iyi performans göstermesi amaçlanmıştır.

Histogram eşitleme ve renk dönüşümleri kullanılarak modelin kontrast ve parlaklık değişimlerine karşı duyarlılığı artırılmıştır. Renk bozulmaları (Color Jittering) yöntemi ile modelin farklı ışık ve renk değişimlerine adapte olması sağlanmıştır. Kenar algılama teknikleri ile modelin şekil ve nesne tanıma yetenekleri geliştirilmiştir. Veri çoğaltma süreci sayesinde, modelin eğitim verisi daha çeşitli hale getirilmiş ve gerçek dünya koşullarında daha yüksek doğrulukta tahmin yapabilecek bir yapıya kavuşturulmuştur.

Veri çoğaltma işlemlerinin otomatik olarak gerçekleştirilmesi için EK B`deki Python programı geliştirilmiştir. Program, belirtilen bir klasördeki görüntüleri alarak, belirlenen veri çoğaltma tekniklerini uygular ve işlenmiş görüntüleri kaydeder.

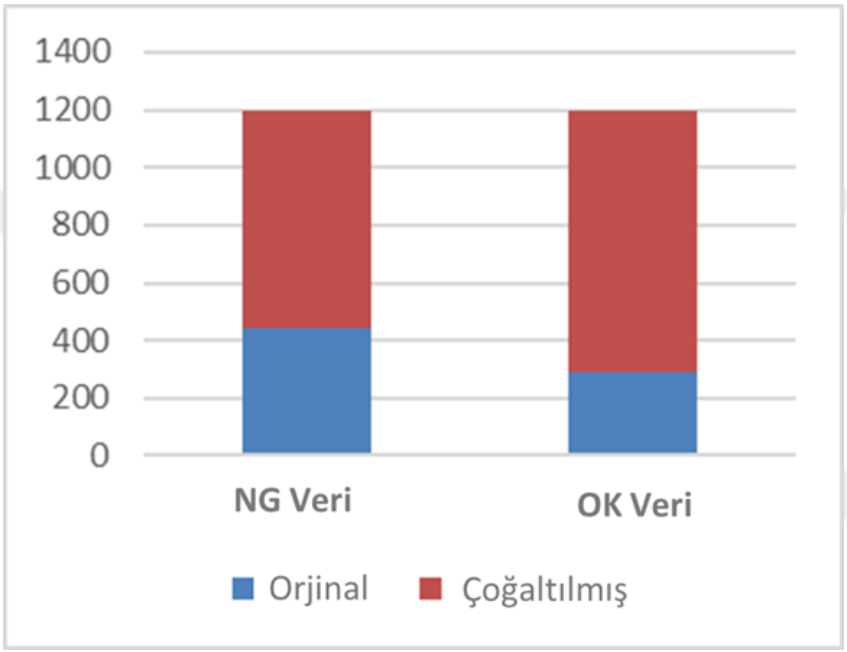
Kodun temel işleyişi şu şekildedir:

1. Kullanıcı tarafından belirtilen klasör taranarak tüm görüntü dosyaları listelenir.
2. Her görüntüye belirlenen veri çoğaltma teknikleri uygulanır.
3. İşlenmiş görüntüler belirli bir isimlendirme formatıyla yeni bir klasöre kaydedilir.
4. Tüm işlemler tamamlandığında yeni veri seti, model eğitimi için kullanıma hazır hale getirilir.

Kod içerisinde farklı veri çoğaltma tekniklerini uygulayan fonksiyonlar bulunmaktadır. apply_histogram_equalization() fonksiyonu, histogram eşitleme işlemi yaparak görüntünün kontrastını artırır. convert_color_space() fonksiyonu görüntüyü HSV renk uzayına dönüştürerek modelin farklı renk spektrumlarına duyarlılığını artırır. normalize_image() fonksiyonu, görüntüleri normalleştirerek modelin öğrenme sürecini hızlandırır. apply_noise_reduction() fonksiyonu, Gaussian veya Median Blur kullanarak gürültü azaltma işlemi gerçekleştirir. apply_edge_detection() fonksiyonu, kenar tespiti yaparak modelin şekil tanıma yeteneğini artırır. add_noise() fonksiyonu, Gaussian veya Tuz & Biber gürültüsü ekleyerek modelin rastgele değişken koşullara adapte olmasını sağlar. apply_color_jitter() fonksiyonu, parlaklık, kontrast ve renk doygunluğu değiştirerek modelin farklı ışık koşullarına uyum sağlamasına yardımcı olur. Geliştirilen kod, farklı veri çoğaltma yöntemlerini dinamik olarak uygulayabilme yeteneğine sahiptir. Bu sayede, modelin ihtiyaç duyduğu veri çeşitliliği sağlanarak eğitim sürecinin daha etkili hale getirilmesi mümkün olmuştur.

Tablo 3.2. Data Çoğaltma Sayıları

No	Yöntem	Varyasyon	Çoğaltılan Veri
1	Görüntü Normalizasyonu	1	5
2	Histogram Eşitleme	1	5
3	Gürültü Azaltma	12	60
4	Kenarlık Algılama	1	5
5	Gürültü Eklemek	28	140
6	Gürültü Eklemek salp_pepper	2	10
7- NG	Renk Bozulmaları	106	530
7-OK		137	686



Şekil.3.7. Veri Dağılımı

Şekil.3.7'de veri dağılımı görülmektedir.

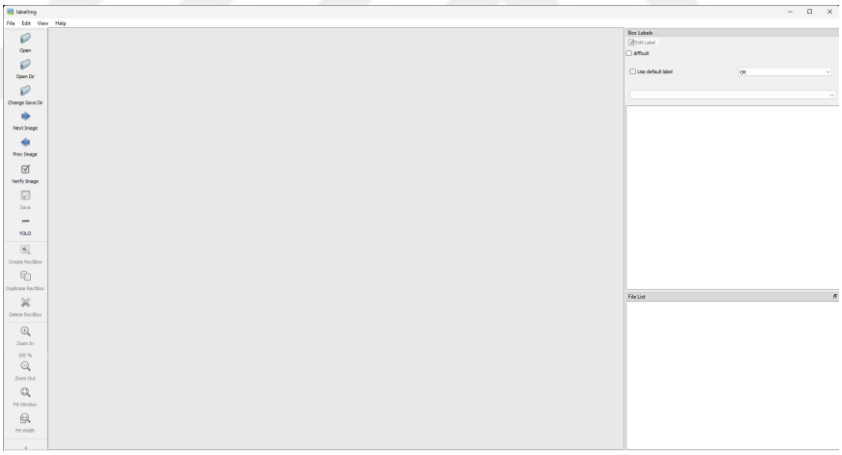
3.7. Veri Etiketleme

Yapay zeka modellerinin başarılı bir şekilde eğitilebilmesi için, modelin öğrenebileceği etiketlenmiş verilere ihtiyaç duyulmaktadır. Bu süreçte, görüntü üzerinde nesnelerin doğru bir şekilde işaretlenmesi ve her nesnenin belirli bir sınıfa atanması gerekmektedir. Bu çalışmada, veri etiketleme işlemi için LabelImg aracı kullanılmıştır. LabelImg, açık kaynaklı bir grafiksel kullanıcı arayüzü (GUI) tabanlı etiketleme aracı olup, manuel olarak görüntü üzerindeki nesneleri işaretlemeye ve bu bilgileri YOLO formatında kaydetmeye olanak tanır. Etiketleme süreci, modelin doğru şekilde eğitilmesi için kritik bir adımdır ve hatalı etiketlemeler modelin performansını olumsuz etkileyebilir.

Veri Etiketleme Süreci:

1. LabelImg aracı açılarak, etiketlenecek görüntülerin bulunduğu klasör seçilir.
2. Etiketleme işlemi için YOLO formatı seçilir.
3. Görüntü üzerinde ilgili nesneler manuel olarak çerçevelenir (Bounding Box).
4. Her nesneye uygun bir sınıf atanır ve etiket kaydedilir.
5. Etiketlenen görüntülerle eşleşen .txt dosyaları oluşturularak, her satırda nesnenin sınıfı ve koordinat bilgileri kaydedilir.
6. Tüm görüntüler tamamlandığında, oluşturulan etiket dosyaları eğitim sürecinde kullanılmak üzere modelin veri setine eklenir.

Etiketleme işlemi tamamlandıktan sonra, etiketlerin doğruluğu kontrol edilerek olası hatalar düzeltilmelidir. Doğru etiketlenmiş veri seti, modelin daha yüksek doğrulukla öğrenmesini sağlar ve yapay zekanın genel performansını artırır. Şekil 3.8'te labelImg'in arayüzü görülmektedir.



Şekil.3.8. LabelImg Arayüz Görseli

3.8. Yapay Zeka Modelinin Eğitilmesi Yolov7-tiny

3.8.1. Yapay zeka modeli için verilerin eğitim ve test olarak ayrılması

Yapay zeka modellerinin başarılı bir şekilde eğitilebilmesi için, verilerin eğitim (training) ve doğrulama (validation) setleri olarak ayrılması gerekmektedir. Eğitim verisi, modelin öğrenme sürecinde kullanılırken, doğrulama verisi, modelin eğitim sırasında öğrenmediği veriler üzerinde test edilerek performansını değerlendirmek için kullanılır. Bu süreç, modelin genelleme yeteneğini artırmak ve aşırı öğrenme

(overfitting) riskini azaltmak için önemlidir. Bu çalışmada, veri seti belirli bir orana göre ikiye ayrılmıştır. Eğitim veri seti genellikle toplam verinin %70-80'ini kapsarken, doğrulama veri seti %20-30'luk bir kısmı oluşturur. Doğru bir eğitim süreci için, her iki veri setinin de benzer dağılıma sahip olması önemlidir.

Veri setini eğitim ve doğrulama olarak ayırmak için EK C`deki Python kodu ile bir fonksiyon geliştirilmiştir. split_dataset_copy() fonksiyonu, .png ve .txt uzantılı dosyaları belirlenen orana göre kopyalayarak ilgili klasörlere ayırmaktadır.

Kodun temel işleyişi şu şekildedir:

1. Eğitim ve doğrulama klasörleri oluşturulur.
    - training/data → Eğitim görüntüleri
    - training/labels → Eğitim etiketleri
    - validation/data → Doğrulama görüntüleri
    - validation/labels → Doğrulama etiketleri
2. Ana klasördeki tüm .png görüntü dosyaları listelenir ve rastgele karıştırılır.
3. Belirtilen eğitim oranına (örneğin %70) göre görüntüler ikiye ayrılır.
4. Eğitim setine ayrılan görüntüler ve onlara ait .txt etiket dosyaları training klasörüne kopyalanır.
5. Doğrulama setine ayrılan görüntüler ve onlara ait .txt etiket dosyaları validation klasörüne kopyalanır.
6. İşlem tamamlandığında, kaç dosyanın eğitim ve doğrulama setlerine ayrıldığı ekrana yazdırılır.

Bu kod, orijinal verileri taşımadan sadece kopyalayarak işlem yaptığı için güvenli bir veri bölme işlemi sağlar. Bu sayede, veri kaybı riski olmadan veri setleri farklı oranlarla yeniden oluşturulabilir.Modelin başarılı bir şekilde eğitilebilmesi için, veri setinin dengeli ve yeterli çeşitliliğe sahip olması önemlidir. Bu fonksiyon sayesinde, eğitim ve doğrulama verileri düzenli bir şekilde ayrılmış ve modelin sağlıklı bir şekilde öğrenmesi için uygun hale getirilmiştir.

3.8.2. Yapay zeka modelini eğitmek için ortam kurulumu

Yapay zeka tabanlı nesne tespiti için kullanılan YOLOv7-tiny modeli, yüksek hız ve doğruluk oranıyla öne çıkmaktadır. Modelin başarılı bir şekilde eğitilebilmesi için uygun bir geliştirme ortamının oluşturulması gerekmektedir. Bu süreçte Python, PyTorch, CUDA, cuDNN ve diğer bağımlılıkların eksiksiz bir şekilde yüklenmesi sağlanmalıdır. Aşağıda, YOLOv7 modelini eğitebilmek için gerçekleştirilen ortam kurulum adımları detaylandırılmıştır.

3.8.2.1. Python ve conda ortamının hazırlanması

YOLOv7'nin eğitimi için Python 3.7 veya 3.8 sürümleri önerilmektedir. Geliştirme ortamı için Miniconda veya Anaconda kullanılabilir. Bu sayede bağımsız bir Python ortamı oluşturularak kütüphane uyumlulukları daha kolay yönetilebilir. Miniconda'nın kurulumu için işletim sistemine uygun sürüm indirilir. Kurulum tamamlandıktan sonra Komut Satırı veya Anaconda Prompt üzerinden bir Conda ortamı oluşturulur ve etkinleştirilir.

3.8.2.2. PyTorch ve CUDA kurulumu

YOLOv7 modeli PyTorch derin öğrenme kütüphanesi üzerine inşa edilmiştir. Bu nedenle, PyTorch'un NVIDIA GPU hızlandırmalı sürümü yüklenmelidir. GPU kullanılmadığında eğitim süreci önemli ölçüde yavaşlayabilir. NVIDIA ekran kartı bulunan sistemlerde GPU desteği için gerekli olan bileşenler CUDA ve cuDNN'dir. Öncelikle, sistemde yüklü olan NVIDIA ekran kartı sürümünün güncel olup olmadığı kontrol edilmelidir. Güncelleme için NVIDIA resmi web sitesi ziyaret edilmelidir. CUDA ve cuDNN kurulumları, NVIDIA'nın resmi web sitesinden uygun sürümler indirilerek gerçekleştirilir. CUDA 11.6 veya 11.7 sürümleri ile cuDNN 8.2 veya 8.4 sürümlerinin uyumlu olduğu test edilmelidir. CUDA ve cuDNN yüklendikten sonra, PyTorch'un CUDA destekli sürümü yüklenmelidir. Conda ortamında ilgili komut çalıştırılarak uyumlu PyTorch ve Torchvision kütüphaneleri yüklenir. Kurulum tamamlandıktan sonra PyTorch'un doğru şekilde çalışıp çalışmadığını test etmek için Python üzerinden GPU kullanılabilirliği kontrol edilir. Eğer doğru şekilde yüklendiyse, sistemde bulunan GPU modeli görüntülenir.

3.8.2.3. YOLOv7-tiny deposu ve bağımlılıkların kurulumu

YOLOv7-tiny modelinin eğitimi için GitHub deposunun klonlanması ve gerekli bağımlılıkların yüklenmesi gerekmektedir. Öncelikle, GitHub'dan YOLOv7 deposu

yerel makineye çekilir ve bağımlılıkların yüklenmesi için requirements.txt dosyası kullanılır. Eğer requirements.txt eksikse, bağımlılıklar manuel olarak yüklenebilir. OpenCV, NumPy, PyYAML, matplotlib, tqdm ve tensorboard gibi kütüphanelerin kurulu olması gerekmektedir.

3.8.2.4. YOLOv7-tiny modelinin eğitilmesi

Kurulum tamamlandıktan sonra, eğitim sürecini başlatmak için modelin eğitim parametreleri belirlenmeli ve uygun bir komut çalıştırılmalıdır. Eğitim sürecinde modelin kaç epoch boyunca çalışacağı, batch size değeri, kullanılacak cihazın GPU veya CPU olup olmadığı, modelin ağırlıklarının nereden yükleneceği gibi ayarlar belirlenmelidir. Model eğitimi sırasında GPU bellek kullanımı izlenmeli ve sistem kaynakları gerektiğinde optimize edilmelidir. Modelin eğitimi tamamlandığında, en iyi ağırlık dosyası kaydedilecek ve test işlemleri için hazır hale getirilecektir. YOLOv7 modelinin eğitimi için uygun bir ortam kurmak, modelin verimli çalışmasını ve hızlı öğrenmesini sağlamak açısından kritik bir adımdır. Bu çalışmada, Miniconda kullanılarak izole bir Python ortamı oluşturulmuş, PyTorch ve CUDA entegrasyonu sağlanmış, YOLOv7'nin GitHub deposu klonlanarak bağımlılıklar yüklenmiş ve modelin eğitime hazır hale getirilmesi için gerekli yapılandırmalar yapılmıştır. Oluşturulan eğitim ortamı, büyük veri setleriyle eğitim yapmaya uygun, GPU hızlandırmalı, tekrarlanabilir ve modüler bir sistem sunmaktadır.

3.9. Yapay Zeka Modelinin Değerlendirilmesi

Bu bölümde, yapay zeka tabanlı nesne tanıma modelinin değerlendirilmesinde kullanılan temel performans ölçütleri tanıtılmaktadır.

3.9.1. Performans değerlendirme kriterleri

Doğruluk (accuracy), duyarlılık (sensitivity), özgüllük (specificity), kesinlik (precision), F1-ölçütü (F1-score) ve ortalama doğruluk değeri (mAP) gibi istatistiksel metrikler, bir modelin sınıflandırma başarısını anlamak için yaygın olarak kullanılan kriterlerdir. Bu metrikler sayesinde, modelin doğru tahmin oranı, hatalı sınıflandırmalara karşı dayanıklılığı ve genel başarımı sayısal olarak değerlendirilebilmektedir.

3.9.1.1. Doğruluk oranı (Accuracy)

Doğruluk oranı, modelin toplam doğru tahminlerinin tüm tahminlere oranını gösterir. Genel model başarımını anlamak için kullanılan en temel metriklerden biridir.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \tag{3.1}$$

Bu oran, modelin genel olarak doğru sınıflandırma yapma yüzdesini temsil eder. Elde edilen yüksek doğruluk oranı, modelin hem pozitif hem de negatif sınıfları başarılı bir şekilde ayırt edebildiğini göstermektedir.

3.9.1.2. Duyarlılık (Recall / Sensitivity)

Duyarlılık, modelin gerçek pozitifleri doğru bir şekilde tespit etme oranını ifade eder. Başka bir deyişle, pozitif sınıfa ait olan örneklerden kaç tanesinin doğru olarak pozitif olarak sınıflandırıldığını gösterir.

$$Sensitivity = \frac{TP}{TP + FN} \tag{4.2}$$

Duyarlılığın yüksek olması, modelin gerçek pozitifleri yakalamada başarılı olduğunu gösterir. Ancak, yanlış negatiflerin yüksek olması modelin bazı gerçek pozitifleri kaçırdığı anlamına gelir. Bu çalışmada, modelin duyarlılık değeri oldukça yüksek çıkmıştır, bu da modelin hatalı negatifleri minimize ettiğini göstermektedir.

3.9.1.3. Özgüllük (Specificity)

Özgüllük, modelin gerçek negatifleri doğru bir şekilde tanımlama oranını gösterir. Yani negatif sınıfa ait olan verilerin ne kadar doğru şekilde negatif olarak tespit edildiğini ölçer.

$$Specificity = \frac{TN}{TN + FP} \tag{4.3}$$

Özgüllüğün yüksek olması, modelin yanlış pozitif tahminlerini düşük tuttuğunu gösterir. Bu çalışmada, özgüllük oranı yüksek olup, modelin yanlış pozitif tahminlerden kaçındığını göstermektedir.

3.9.1.4. Kesinlik (Precision)

Kesinlik, modelin pozitif tahminlerinin doğruluğunu gösterir. Yani model tarafından pozitif olarak tahmin edilen örneklerden kaçarının gerçekten pozitif olduğunu ölçer.

$$Precision = \frac{TP}{TP + FP} \quad (4.4)$$

Kesinlik oranının yüksek olması, modelin yanlış pozitif oranının düşük olduğunu ve modelin hatalı pozitif tahmin yapmaktan kaçındığını gösterir. Bu çalışmada, kesinlik değeri oldukça yüksek olup, modelin yanlış pozitif oranının düşük olduğunu göstermektedir.

3.9.1.5. F1-Ölçümü (F1-Score)

F1-ölçümü, duyarlılık (recall) ve kesinlik (precision) arasındaki dengeyi ölçen bir metriktir. Bu değer, özellikle veri dengesizliği olduğu durumlarda önemli bir kriterdir. Duyarlılığı ve kesinliği aynı anda optimize eden bir değer sunar.

$$F1 - Score = 2 \text{ x } \frac{Precision \ x \ Sensitivity}{Precision + Sensitivity} \quad (4.5)$$

F1-ölçümünün yüksek olması, hem yanlış pozitiflerin hem de yanlış negatiflerin düşük olduğu anlamına gelir. Bu çalışmada, modelin F1-ölçümü yüksek bir değere sahiptir ve modelin pozitif sınıfları tespit etme konusunda oldukça dengeli bir performans gösterdiğini kanıtlamaktadır.

3.9.1.6. Confusion matrix ve model performans analizi

Confusion Matrix, modelin sınıflandırma hatalarını analiz etmek için kullanılan bir tablo olup, modelin doğru ve yanlış tahminlerini dört farklı kategoriye ayırır:

- True Positives (TP): Modelin doğru bir şekilde pozitif olarak tahmin ettiği örnekler.
- False Positives (FP): Modelin yanlış bir şekilde pozitif olarak tahmin ettiği (aslında negatif olan) örnekler.
- False Negatives (FN): Modelin yanlış bir şekilde negatif olarak tahmin ettiği (aslında pozitif olan) örnekler.
- True Negatives (TN): Modelin doğru bir şekilde negatif olarak tahmin ettiği örnekler.

Confusion Matrix, modelin ne kadar doğru tahmin yaptığını görsel olarak incelemek için kullanılır. TP ve TN değerleri yüksekse, modelin genel doğruluğu yüksektir. FP ve FN değerlerinin düşük olması ise modelin yanlış tahmin oranlarının düşük olduğunu gösterir.

3.9.1.7. Kutu kaybı (Box Loss)

Box loss, modelin tespit ettiği nesne kutularının, gerçek kutularla ne derece örtüştüğünü ölçen bir değerdir. Bu kayıp fonksiyonu, tespit edilen dikdörtgenlerin konum ve boyutlarının doğruluğunu optimize etmeye yöneliktir. Düşük box loss değeri, modelin nesnelerin konumlarını daha doğru tespit ettiğini gösterir.

3.9.1.8. Nesne varlığı kaybı (Objectness Loss)

Objectness loss, modelin görüntüde gerçekten bir nesne olup olmadığını doğru şekilde belirleyebilme yeteneğini ölçer. Bu metrik, modelin arka planı ve ilgisiz alanları ayırt edebilme başarısını gösterir. Düşük değer, modelin yalnızca gerçek nesneleri dikkate aldığını ve yanlış pozitifleri azalttığını gösterir.

3.9.1.9. Sınıflandırma kaybı (Classification Loss)

Classification loss, modelin tespit ettiği nesneleri doğru sınıfa atama yeteneğini değerlendirir. Bu metrik, nesnelerin doğru etiketlerle sınıflandırılıp sınıflandırılmadığını belirler. Düşük sınıflandırma kaybı, modelin tahminlerinin etiket bazında doğruluğunun yüksek olduğunu ifade eder.

3.9.1.10. Ortalama doğruluk (mAP@0.5)

mAP@0.5, tahmin edilen kutuların, gerçek kutularla en az %50 örtüşmesi koşuluyla hesaplanan ortalama doğruluk değeridir. Bu metrik, modelin nesne tespiti başarımını tek bir eşik değeri üzerinden değerlendirir ve genel tespit performansını özetler.

3.9.1.11. Genişletilmiş ortalama doğruluk (mAP@0.5:0.95)

mAP@0.5:0.95, farklı örtüşme eşik değerleri (%50'den %95'e kadar) kullanılarak hesaplanan ortalama doğruluk oranıdır. Bu metrik, modelin farklı hassasiyet düzeylerindeki başarısını çok daha detaylı bir biçimde yansıtır. Yüksek değerler, modelin yalnızca genel başarıyı değil, aynı zamanda zorlu senaryolarda da etkili olduğunu gösterir.

3.10. Sistem Yazılımının Geliştirilmesi ve Yapay Zeka Modelinin Entegrasyonu

Sistemin yazılım geliştirme süreci, modüler bir yaklaşımla ele alınmış ve her bileşen belirli görevleri yerine getirecek şekilde tasarlanmıştır. Bu yaklaşım, sistemin yönetilebilirliğini artırarak hata ayıklama sürecini kolaylaştırmış ve geliştirme sürecini

daha esnek hale getirmiştir. Geliştirilen yazılım bileşenleri aşağıdaki adımlara göre yapılandırılmıştır:

1- Kamera Görüntü İşleme Modülü :

Sistemin temel bileşenlerinden biri, kameradan görüntü almayı sağlayan yazılım modülüdür. Bu modül, Basler marka kameralar ile çalışacak şekilde geliştirilmiş olup, görüntülerin gerçek zamanlı olarak alınmasını ve işlenmesini sağlamaktadır. Kamera bağlantısı sağlandıktan sonra belirli aralıklarla görüntü alınarak, yapay zeka modeline gönderilmek üzere işlenmektedir.

2- PLC ile Modbus Haberleşme Modülü

Sistem, üretim hattındaki belirli işlemleri koordine edebilmek ve dış dünyadan veri alabilmek için bir PLC (Programmable Logic Controller) ile haberleşmektedir. Bu haberleşmeyi sağlamak amacıyla, Adam-IO cihazı üzerinden Modbus protokolü kullanılarak tetikleme verileri alınmaktadır. PLC'den gelen sinyaller, sistemin ne zaman çalışması gerektiğini belirleyerek görüntü işleme sürecinin doğru zamanda başlamasını sağlamaktadır.

3- Yapay Zeka Modeli ile Nesne Tespit Modülü

Elde edilen görüntülerin analizi, YOLOv7-tiny modelinin eğitilerek kullanılmasıyla gerçekleştirilmiştir. Bu amaçla geliştirilen **AI Model** yazılım modülü, alınan görüntüleri işleyerek nesne tespiti yapmakta ve belirlenen sınıflara göre bir değerlendirme sonucu üretmektedir. Modelin gerçek zamanlı çalışmasını sağlamak amacıyla optimize edilmiş bir yapı benimsenmiş ve işlem süresinin en aza indirilmesi hedeflenmiştir.

4- Kullanıcı Arayüzü ve Görselleştirme Modülü

Model tarafından işlenen görüntülerin kullanıcıya sunulabilmesi için **PyQt5** kütüphanesi kullanılarak bir arayüz geliştirilmiştir. Bu arayüz, sistemin aldığı görüntüleri ve modelin tespit sonuçlarını görselleştirmek amacıyla tasarlanmıştır. Kullanıcı, arayüz üzerinden anlık olarak görüntüleri inceleyebilir ve sistemin verdiği kararları takip edebilir.

5- Ana Yazılım Entegrasyonu

Geliştirilen tüm bu yazılım modülleri, main.py adlı ana dosya altında birleştirilmiştir. Böylece her bileşenin kendi görevini yerine getirdiği modüler

bir yapı oluşturulmuş ve tüm sistemin tek bir merkezden yönetilmesi sağlanmıştır. Bu yapı sayesinde sistemin ölçeklenebilirliği artırılmış, gerektiğinde yeni bileşenlerin eklenmesine olanak tanınmıştır.

Bu yazılım mimarisi, yapay zeka tabanlı görüntü işleme süreçlerinin etkin bir şekilde yürütülmesini sağlamakta ve üretim ortamlarında uygulanabilir bir sistem sunmaktadır.

3.10.1. Kamera işleme modülü

Bu bölümde, Basler marka kameradan görüntü almak için geliştirilen yazılımın işleyişi detaylandırılmaktadır.

EK D`de gelistirilen kodda kamera ile etkileşimi sağlamak, görüntü almak ve sistemin performansını optimize etmek amacıyla Pypylon kütüphanesi kullanılmıştır. Kod, çoklu iş parçacığı (threading) yapısı ile çalışarak kamera ile bağımsız bir süreçte haberleşmeyi mümkün kılmaktadır. Böylece sistemin genel performansını artırmak ve görüntü işlemenin stabil çalışmasını sağlamak hedeflenmiştir.

Kod, CameraThread adlı bir sınıf üzerinden çalışmaktadır. Bu sınıf, Python'un Thread modülünü miras alarak kendi başına bir iş parçacığı olarak çalışabilme özelliğine sahiptir. Kamera başlatıldığında ayrı bir thread olarak çalışmaya başlar ve sürekli olarak görüntü almak yerine, sadece belirli bir fonksiyon çağrıldığında görüntü yakalar. Bu yöntem, gereksiz kaynak tüketimini önleyerek sistemin verimli çalışmasını sağlamaktadır.

Sistemin başlatılması sırasında log mekanizması devreye alınarak, her önemli adım ve olası hata kayıt altına alınmaktadır. Python'un logging kütüphanesi ile oluşturulan bu yapı, uygulamanın çalışma durumunu izlemeye ve hata ayıklamaya yardımcı olmaktadır. Log verileri, app_basler_camera.log dosyasında saklanmaktadır ve anlık olarak konsola da yazdırılmaktadır.

Kodun temel işleyişi aşağıdaki gibi özetlenebilir:

- **Kamera Başlatma :**Kamera **pylon.TlFactory.GetInstance().Create FirstDevice()** komutu ile sistemde tanımlı ilk Basler cihazını kullanacak şekilde başlatılmaktadır. Kamera başarılı bir şekilde başlatıldığında, görüntüleri yakalamak için **GrabStrategy_LatestImageOnly** stratejisi

uygulanmaktadır. Bu strateji, en son alınan görüntünün sürekli olarak kullanılmasını sağlayarak, sistemin güncel verilerle çalışmasını mümkün kılar.

- **Çalışma Mantığı :run()** fonksiyonu, kamera başlatıldığında çalışmaya başlar. Ancak, gereksiz işlem gücü tüketimini önlemek adına sürekli görüntü almak yerine, sistemin belirli aralıklarla beklemesini sağlamaktadır. Böylece, kamera açık kalırken sadece get_frame() fonksiyonu çağrıldığında görüntü alınacaktır.
- **Görüntü Alımı :get_frame()** fonksiyonu, kameradan tek bir kare görüntü almak için kullanılır. Görüntü yakalandığında, pylon.ImageFormat Converter aracılığıyla BGR formatına dönüştürülerek kullanılabilir hale getirilir. Çıktı, thread güvenliği sağlamak adına Lock nesnesi kullanılarak korunur ve başka bir süreç tarafından erişilirken veri bütünlüğü korunur.
- **Thread ve Kamera Kapatma:**stop() fonksiyonu, kamera bağlantısını sonlandırmak ve thread sürecini güvenli bir şekilde durdurmak için kullanılır. Kamera hala aktif bir şekilde görüntü alıyorsa, StopGrabbing() metodu çağrılarak işlem tamamlanır. Kamera kapatıldığında, sistemin stabil çalışmasını sağlamak amacıyla log mekanizması kullanılarak bilgi mesajı kaydedilir.

Bu yapı, modüler bir şekilde çalışarak sistemin diğer bileşenleriyle kolayca entegre olmasını sağlamaktadır. Kamera, gereksiz işlem yükü oluşturmadan sürekli açık kalabilir ve gerektiğinde tekil görüntüler alınarak işlemeye devam edebilir. Böylece, endüstriyel uygulamalarda gerçek zamanlı ve yüksek performanslı bir görüntü işleme altyapısı oluşturulmuştur.

3.10.2. PLC ile modbus haberleşme modülü

Bu çalışmada, **Modbus TCP/IP** protokolü kullanılarak PLC ile haberleşme sağlanmış ve sistemin ihtiyaç duyduğu veri alışverişi gerçekleştirilmiştir. Geliştirilen yazılım, pymodbus kütüphanesi kullanılarak Python programlama dili ile oluşturulmuştur. Haberleşme modülü, ModbusTcpClient sınıfı aracılığıyla PLC'ye bağlanarak bobin (coil), giriş (input register), tutma (holding register) ve ayrık giriş (discrete input) verilerini okuma ve yazma işlemlerini gerçekleştirebilmektedir.

EKLER E'de, PLC ile haberleşmek amacıyla geliştirilen kod sunulmuştur.

- **Modülün Başlatılması:**Modbus haberleşme modülü, ModbusTcpClient sınıfı kullanılarak bir TCP/IP bağlantısı kurarak başlatılmaktadır. ModbusTcpClient(self.TCP_IP, self.TCP_PORT) komutu, belirtilen IP adresi ve port üzerinden PLC ile iletişimi sağlar. Başlatma işlemi sırasında, bağlantının başarılı olup olmadığı try-except yapısı ile kontrol edilmekte ve hata oluşması durumunda sistem loglarına kaydedilmektedir.
- **Çalışma Mantığı:** Run() fonksiyonu, haberleşme modülü başlatıldığında aktif hale gelir. Ancak sistem performansını optimize edebilmek adına, sürekli veri çekmek yerine, ihtiyaç duyulan anlarda belirli kayıt türlerinden okuma ve yazma işlemleri gerçekleştirilir. readCoilsRegister(), readHoldingRegister() gibi fonksiyonlar, sistemin ilgili kayıt noktalarına erişmesini sağlar.
- **Modbus Kayıtlarının Yönetimi:** Coil (Bobin) Okuma ve Yazma: readCoilsRegister() fonksiyonu, PLC'nin dijital çıkışlarını okuyarak sistemin güncel durumu takip etmesini sağlar. writeCoils() fonksiyonu ise belirli bobinlere yazma işlemi gerçekleştirerek kontrol mekanizmasını oluşturur.Input Register (Giriş Kayıtları) Okuma: readInputRegister() fonksiyonu, PLC'nin sensörlerden aldığı giriş değerlerini okumak için kullanılır.Holding Register (Tutma Kayıtları) Okuma: readHoldingRegister() fonksiyonu, PLC'nin uzun süreli hafızasında tutulan verileri okuyarak analiz edilmesini sağlar.Discrete Inputs (Ayrık Girişler) Okuma: readDiscreteInputs() fonksiyonu, dijital sensörlerden gelen bilgileri sisteme kazandırmak için kullanılır.
- **Bağlantının Sonlandırılması:** Sistem kapatılmak istendiğinde, closeConnection() fonksiyonu çağrılarak PLC ile olan bağlantı güvenli bir şekilde sonlandırılır. Bu işlem, haberleşme sırasında oluşabilecek hataları önlemek için gereklidir.

3.10.3. Yapay zeka modeli ile nesne tespit modülü

Geliştirilen nesne tespit modülü, YOLOv7-tiny modelini kullanarak görüntülerdeki nesneleri tespit etmek ve sınıflandırmak amacıyla tasarlanmıştır. Bu modül, önceden eğitilmiş bir derin öğrenme modelini kullanarak endüstriyel denetim sistemlerinde doğrulukla çalışmasını sağlamak için optimize edilmiştir.Söz konusu modüle ait detaylı Python kodu EKLER F'de sunulmuştur.

Nesne tespit modülü, gelen görüntüleri işleyerek belirlenen sınıflar doğrultusunda nesne tespiti yapar. Sistem, CUDA destekli GPU kullanımı ile hızlandırılabilmekte olup, GPU'nun mevcut olmadığı durumlarda CPU üzerinde de çalışabilmektedir.

Modülün temel çalışma aşamaları şu şekildedir:

- **Modelin Yüklenmesi:** YOLOv7-tiny modeli, sistemin mevcut donanımına uygun olarak yüklenir ve kullanıma hazır hale getirilir. Model, GPU destekli sistemlerde CUDA kullanılarak çalıştırılır, ancak GPU bulunmadığında otomatik olarak CPU modunda çalışacak şekilde yapılandırılmıştır.
- **Görüntü İşleme:** Tespit edilecek görüntü, modelin giriş formatına uygun hale getirilmek için ön işleme tabi tutulur. **Letterbox yöntemi** kullanılarak görüntü 640x640 boyutuna orantılı şekilde yeniden ölçeklendirilir. Görüntü, YOLOv7'nin gereksinimlerine uygun şekilde RGB renk formatına çevrilir ve tensör formuna dönüştürülerek modele giriş olarak verilir.
- **Model Çıktısının İşlenmesi:** Model, görüntü üzerindeki nesneleri tespit ettikten sonra **Non-Maximum Suppression (NMS)** uygulanarak gereksiz ve çakışan kutular elenir. Elde edilen sonuçlar, sınıf etiketi, güven skoru ve nesnenin koordinatları ile birlikte kullanıma sunulur.
- **Bounding Box Çizimi ve Görselleştirme:** Tespit edilen nesneler, sınıf etiketleri ve güven skoru ile birlikte görüntü üzerine kutu çizilerek işlenir. NG (Hatalı) ve OK (Doğru) sınıfları için farklı renk kodları kullanılarak nesneler belirgin hale getirilir.
- **Tespit sonuçlarının saklanması:**Sistem, analiz edilen görüntüleri tespit edilen nesne sınıfına bağlı olarak ilgili klasörlere kaydetmektedir. OK olarak sınıflandırılan görüntüler ayrı bir klasörde, NG olarak sınıflandırılan hatalı nesneler farklı bir klasörde arşivlenir. Eğer herhangi bir nesne tespit edilmezse, görüntü NULL kategorisine eklenir ve ilgili log kaydı tutulur.NG (Hatalı) Görüntüler: Belirlenen hatalı nesneler için görüntü, NG klasörüne kayıt edilir.OK (Doğru) Görüntüler: Doğru tespit edilen nesneler OK klasörüne kayıt edilerek ayrıca küçük boyutta bir kopyası oluşturulur. Hiç Nesne Tespit Edilmeyen Görüntüler: Eğer model herhangi bir nesne tespit etmezse, bu görüntü NULL klasörüne kayıt edilerek sistemin analiz sürecinde eksik tespit edilen durumların değerlendirilmesine yardımcı olur.

3.10.4. Kullanıcı arayüzü ve görselleştirme modülü

Bu bölümde, geliştirilen yapay zeka tabanlı nesne tespit sisteminin kullanıcı arayüzü ve görselleştirme modülü açıklanmaktadır. Kullanıcı arayüzü, **PyQt5** kütüphanesi kullanılarak geliştirilmiş olup, modelin tespit ettiği sonuçları gerçek zamanlı olarak kullanıcıya sunmaktadır. Sistemin bu işlevlerini gerçekleştiren kullanıcı arayüzüne ait detaylı kod içeriği EKLER G'de verilmiştir.

Kullanıcı arayüzü, kamera görüntülerinin ve yapay zeka modelinin tespit ettiği sonuçların anlık olarak gösterilmesini sağlarken, sistemin çalışmasını kolaylaştıran tam ekran modu, hata tespiti, renk kodlu sonuç gösterimi gibi fonksiyonları içermektedir. Ana arayüz, üç temel bileşenden oluşmaktadır:

- **Görüntüleme Alanı**: Tespit edilen nesnelerin gösterildiği ana alan burasıdır. Sistemin aldığı en son görüntü burada gösterilir ve modelin yaptığı değerlendirmeye göre görüntü üzerine işlenen sonuçlar yansıtılır. Kullanıcı, tam ekran moduna geçebilir ve görüntüleme alanı otomatik olarak ölçeklenir.
- **Sonuç Gösterim Alanı:** Model tarafından yapılan sınıflandırmanın sonucunu belirten bir bölüm bulunmaktadır. Tespit edilen nesne OK (doğru) veya NG (hatalı) olarak değerlendirildiğinde, arayüzdeki gösterge yeşil veya kırmızı renge dönerek durumu belirtir.
- **Görüntü İşleme ve Güncellenmesi:**Arayüz, anlık olarak modelin sonuçlarını yansıtabilmek için bir görüntü güncelleme mekanizmasına sahiptir. Sistem, belirli bir dizinde depolanan güncel görüntüyü arayüze yükleyerek, kullanıcının en son tespit edilen nesneyi ve yapılan değerlendirmeyi görmesini sağlar. Görüntü boyutları arayüzün genişliğine ve yüksekliğine dinamik olarak ölçeklendirilerek ekrana tam oturtulmaktadır. Letterbox yöntemiyle yapılan ölçekleme, görüntünün bozulmasını önleyerek orantıyı korumaktadır.

3.10.5. Ana yazılım entegrasyonu

Yapay zeka tabanlı nesne tespit sisteminin farklı bileşenlerinin bir araya getirilerek tam entegre bir şekilde çalışmasını sağlamak amacıyla ana yazılım entegrasyonu gerçekleştirilmiştir. Bu entegrasyon, kamera kontrolü, PLC'den tetik alarak görüntü işleme sürecini başlatma, yapay zeka modelinin nesne tespiti yapması, sonuçların görselleştirilmesi ve kaydedilmesi gibi adımlardan oluşmaktadır. Tüm bileşenlerin birlikte çalışmasını sağlayan ana yazılım kodu EKLER H'de yer almaktadır.Ana

yazılım ile sistem bileşenlerinin entegrasyonu aşağıdaki bileşenlerin uyumlu bir şekilde çalışmasını sağlamak üzere geliştirilmiştir:

- Kamera Modülü: Kamera kontrolünü sağlayan CameraThread sınıfı kullanılarak sistemin gerçek zamanlı görüntü alması sağlanmıştır.
- PLC Haberleşme Modülü: Modbus protokolü üzerinden ModbusTcpClient ile PLC'den tetik bilgisi alınarak sistemin belirli olaylara tepki vermesi sağlanmıştır.
- Yapay Zeka Algoritması: YOLOv7-tiny modelini içeren AI modülü, alınan görüntüler üzerinde nesne tespiti yaparak OK veya NG sınıflandırmasını gerçekleştirmektedir.
- Kullanıcı Arayüzü: PyQt5 kullanılarak geliştirilen arayüz, tespit edilen nesneleri ve sınıflandırma sonuçlarını görselleştirmek için tasarlanmıştır.
- Sonuçların Kaydedilmesi: Tespit edilen nesneler ve model çıktıları Excel formatında kaydedilerek veri analizi ve model performans değerlendirmesi için arşivlenmektedir.

Sistemin ana iş akışı PLC'den gelen tetik sinyali ile çalışmaya başlamakta ve aşağıdaki adımları takip etmektedir:

- PLC'den Tetik Algılama Sistem, Modbus haberleşmesi ile PLC'den gelen sinyalleri sürekli olarak izlemektedir. Eğer tetik sinyali (coil register) 1 değerine ulaştığında, sistem yeni bir işlem başlatmaktadır.
- Kamera ile Görüntü Alma :Kamera modülü, kamera donanımını kontrol eden CameraThread sınıfı ile entegre edilmiştir. Gelen tetik sinyali doğrultusunda, anlık bir görüntü alınarak yerel belleğe kaydedilmektedir. Görüntü PNG formatında kaydedilerek, yapay zeka modeli tarafından işlenmeye hazır hale getirilmektedir.
- Yapay Zeka ile Nesne Tespiti :YOLOv7-tiny modeli, alınan görüntüyü analiz ederek NG veya OK sınıflandırması yapmaktadır. Modelin ürettiği güven değeri (confidence score) de kaydedilmektedir. Sınıflandırma sonuçları anlık olarak kullanıcı arayüzüne iletilmektedir.
- Sonuçların Arayüzde Gösterilmesi: PyQt5 tabanlı kullanıcı arayüzü, model tarafından yapılan sınıflandırma sonuçlarını gerçek zamanlı olarak

güncellemektedir. Görüntü alanında en son işlenen görsel gösterilirken, sınıflandırma sonucu OK (yeşil) veya NG (kırmızı) olarak belirtilmektedir.

- Sonuçların Excel Dosyasına Kaydedilmesi: Yapay zeka modelinin tahmin sonuçları ve güven değerleri, zaman damgası ile birlikte Excel dosyasına yazılmaktadır. Bu kayıt mekanizması, sistem performansını takip etmek ve daha sonra analiz edebilmek için geliştirilmiştir.
- Sistem Performansını Artırmak İçin Çalışma Mantığı: Ana işlem döngüsü, mainJob fonksiyonunda çalıştırılmaktadır.

Yapay zeka modeli, sadece gerektiğinde yüklenerek gereksiz işlem yükü önlenmiştir.

Gereksiz işlem gücü tüketimini engellemek için döngüler belirli zaman aralıklarında çalışacak şekilde optimize edilmiştir.

4. SONUÇLAR

Bu çalışmada, geliştirilen YOLOv7-tiny tabanlı yapay zeka modelinin performansını değerlendirmek amacıyla çeşitli istatistiksel metrikler kullanılmıştır. Modelin doğruluğunu, yanlış pozitif ve yanlış negatif oranlarını, güvenilirliğini ve genel başarım seviyesini analiz etmek için doğruluk oranı (accuracy), duyarlılık (sensitivity/recall), özgüllük (specificity), kesinlik (precision) ve F1-ölçümü (F1-score) gibi ölçütler hesaplanmıştır. Bu metrikler, modelin gerçek dünya uygulamalarında ne kadar güvenilir çalıştığını ortaya koymaktadır. Modelin sınıflandırma performansını değerlendirmek için Confusion Matrix (Karışıklık Matrisi) oluşturulmuş ve aşağıdaki temel performans ölçütleri hesaplanmıştır.

4.1. Yapay Zeka Modeli Performans Değerlendirmesi

YOLOv7-tiny modeliyle yapılan eğitim sürecinin kapsamlı bir değerlendirmesi yer almaktadır. Modelin performans analizini yaparken doğruluk, duyarlılık, kesinlik, F1-score, mAP ve hata analizi gibi önemli metrikler dikkate alınmıştır. Ek olarak, eğitim sırasında kullanılan veri artırma yöntemleri ve modelin endüstriyel uygulanabilirliği de ayrıntılı bir şekilde ele alınmıştır.

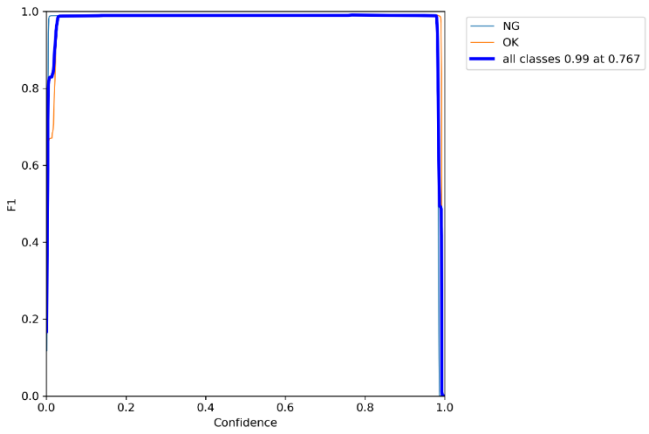
4.1.1. Model eğitim performansının değerlendirilmesi

Bu bölümde, geliştirilen YOLOv7-tiny tabanlı yapay zeka modelinin eğitim süreci boyunca gösterdiği performans ayrıntılı bir şekilde değerlendirilmiştir. Eğitim sürecinin her bir evresinde kaydedilen metrikler, modelin öğrenme düzeyini, genelleme yeteneğini ve doğruluk seviyesini analiz etmek amacıyla kullanılmıştır. Eğitim ve doğrulama kayıplarının yanı sıra, doğruluk (precision), duyarlılık (recall) ve ortalama doğruluk oranı (mAP) gibi performans göstergeleri zamanla takip edilmiş ve grafiklerle görselleştirilmiştir.

4.1.1.1. F1-Skor

F1-score metriği 0.99 seviyesinde olup, modelin duyarlılık ve kesinlik dengesinin oldukça iyi olduğunu göstermektedir. Bu metrik, modelin hem precision (kesinlik) hem de recall (duyarlılık) açısından dengeli bir performans sergilediğini ifade

etmektedir. Şekil 3.17'de farklı güven seviyelerine göre F1-score değerinin nasıl değiştiği gösterilmektedir.

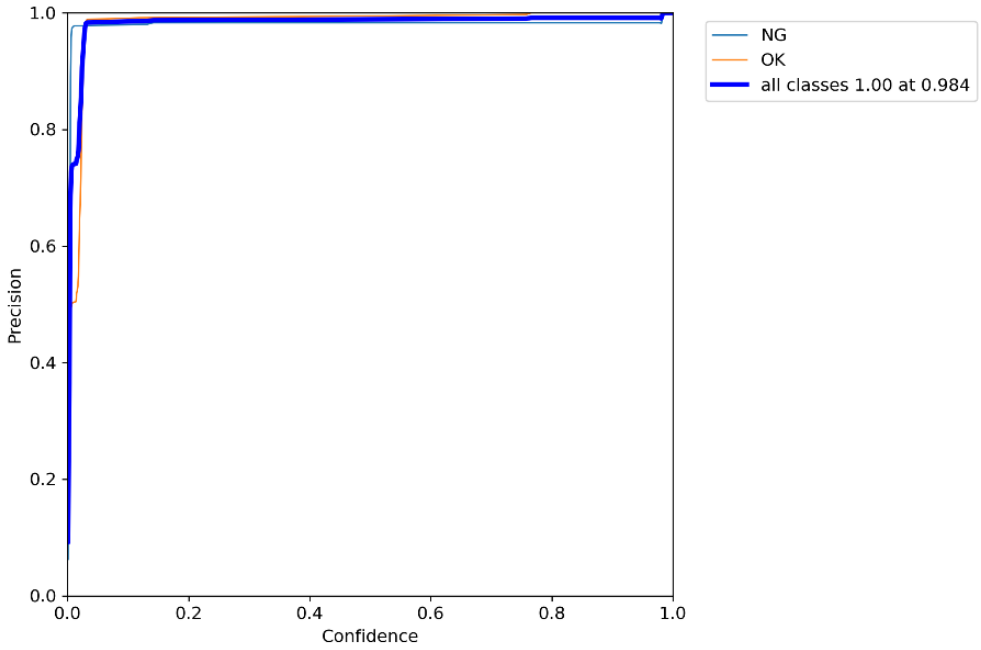


Şekil 4.1. F1-Score ve Confidence Grafiği

Eğrinin erken bir noktada hızla 1.0 seviyesine ulaşması, modelin düşük güven eşiklerinde dahi oldukça başarılı tahminler yapabildiğini göstermektedir. Ayrıca, NG (hatalı) ve OK (doğru) sınıfları için hesaplanan F1-score değerlerinin birbirine yakın olması, modelin her iki sınıfı da tutarlı bir şekilde ayırt edebildiğini göstermektedir. Modelin hatalı pozitif (False Positive) ve hatalı negatif (False Negative) tahminleri düşük seviyede tutması, endüstriyel uygulamalarda güvenilir bir şekilde kullanılabileceğini kanıtlamaktadır. Yüksek F1-score, modelin yanlış tespitlerden kaynaklanan hatalı karar alma olasılığını en aza indirdiğini ve hatalı parçaların doğru parçalarla karışmasının önüne geçtiğini göstermektedir.

4.1.1.2. Precision (Kesinlik)

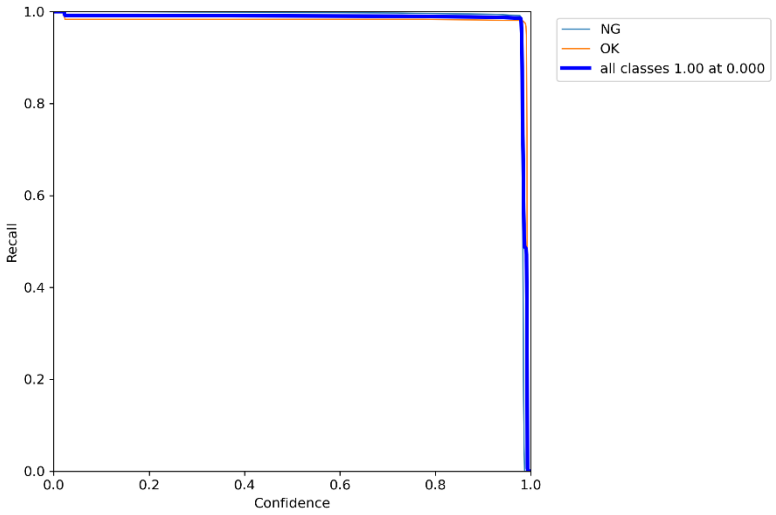
Şekil 3.18'de gösterildiği üzere Precision (Kesinlik) metriği 1.00 seviyesine ulaşmıştır. Precision, modelin tespit ettiği pozitif örnekler arasında gerçekten doğru olanların oranını ifade etmektedir. Yüksek bir precision değeri, modelin yanlış pozitif (False Positive) tahminlerini en aza indirdiğini ve yalnızca gerçekten ilgili nesneleri belirlediğini göstermektedir.



Şekil 4.2. Precision – confidence grafiği

4.1.1.3. Recall (Duyarlılık)

Recall (Duyarlılık) metriği ise 0.99 olup, modelin gerçek pozitifleri yakalama oranının oldukça yüksek olduğunu ortaya koymaktadır. Şekil 3.19'da recall – confidence görülmektedir.



Şekil 4.3. Recall – confidence grafiği

4.1.1.4. Epoch bazında kayıp fonksiyonu analizi

Modelin eğitim sürecinde farklı epoch aralıklarında elde edilen kayıp (loss) değerleri tablo 4.2`de gösterilmiştir. Modelin öğrenme kapasitesinin zamanla arttığı gözlemlenmiştir. Eğitim süreci boyunca takip edilen üç temel kayıp türü olan kutu kaybı (box loss), nesne kaybı (objectness loss) ve sınıflandırma kaybı (classification loss) değerleri, her geçen epoch aralığında azalarak modelin daha doğru tahminler yapabildiğini göstermiştir. Başlangıçta 0.042 seviyesinde olan kutu kaybı, 100. epoch sonunda 0.011'e kadar düşmüş, bu da modelin nesneleri konumlandırma konusunda giderek daha başarılı hale geldiğini ortaya koymuştur. Benzer şekilde, nesne kaybı değeri 0.015'ten 0.0025'e, sınıflandırma kaybı ise 0.017'den 0.0035'e gerileyerek modelin hem nesne varlığını tespit etme hem de doğru sınıflandırma yapma becerisinde önemli bir gelişim sağladığını göstermiştir.

Tablo 4.1. Model eğitim sürecinin kayıp değerlerinin analizi

Epoch Aralığı	Kutu Kayıp (Box Loss)	Nesne Kayıp (Objectness Loss)	Sınıflandırma Kayıp (Classification Loss)
1-10	0.042	0.015	0.017
11-20	0.035	0.012	0.015
21-30	0.028	0.009	0.012
31-40	0.024	0.0075	0.01
41-50	0.02	0.006	0.0085
51-60	0.017	0.005	0.007
61-70	0.015	0.0042	0.006
71-80	0.013	0.0035	0.0048
81-90	0.012	0.003	0.004
91-100	**0.011**	**0.0025**	**0.0035**

4.1.1.5. Epoch bazında keskinlik, duyarlılık ve map değerlerinin değerlendirilmesi

Modelin eğitim süreci, yalnızca kayıp değerlerinin analiziyle değil, aynı zamanda performans metriklerinin gelişimiyle de kapsamlı şekilde değerlendirilmiştir. Tablo 4.3`de performans analizi sonuçları gösterilmiştir. Epoch aralıklarına göre hesaplanan precision (Keskinlik), recall (Duyarlılık), mAP@0.5 ve mAP@0.5:0.95 değerleri, modelin doğruluk ve genel başarı seviyesinin zamanla arttığını açıkça ortaya

koymaktadır. Eğitimin ilk 10 epoch'luk döneminde precision değeri 0.80, recall ise 0.75 olarak ölçülmüştür. Bu erken aşamada model, sınırlı veri öğrenimiyle birlikte makul bir başlangıç performansı göstermektedir. Ancak bu metrikler, eğitim ilerledikçe düzenli bir şekilde artış göstermiştir. Örneğin, 31-40 epoch aralığında precision 0.94'e, recall ise 0.90'a yükselmiş; bu da modelin doğru pozitifleri tespit etme ve yanlış pozitiflerden kaçınma becerisinin önemli ölçüde geliştiğini göstermektedir.mAP (mean Average Precision) değerleri, modelin genel sınıflandırma başarısını yansıtmaktadır. mAP@0.5 değeri ilk 10 epoch'ta 0.78 iken, bu oran 91-100 epoch aralığında 0.985'e kadar çıkmıştır. mAP@0.5:0.95 gibi daha zorlu bir ölçümde bile, eğitim sonunda 0.96'ya ulaşılmıştır. Bu, modelin farklı eşik değerlerinde dahi yüksek doğrulukla tahmin yapabildiğini ve çeşitli senaryolarda sağlam bir performans sergilediğini göstermektedir.

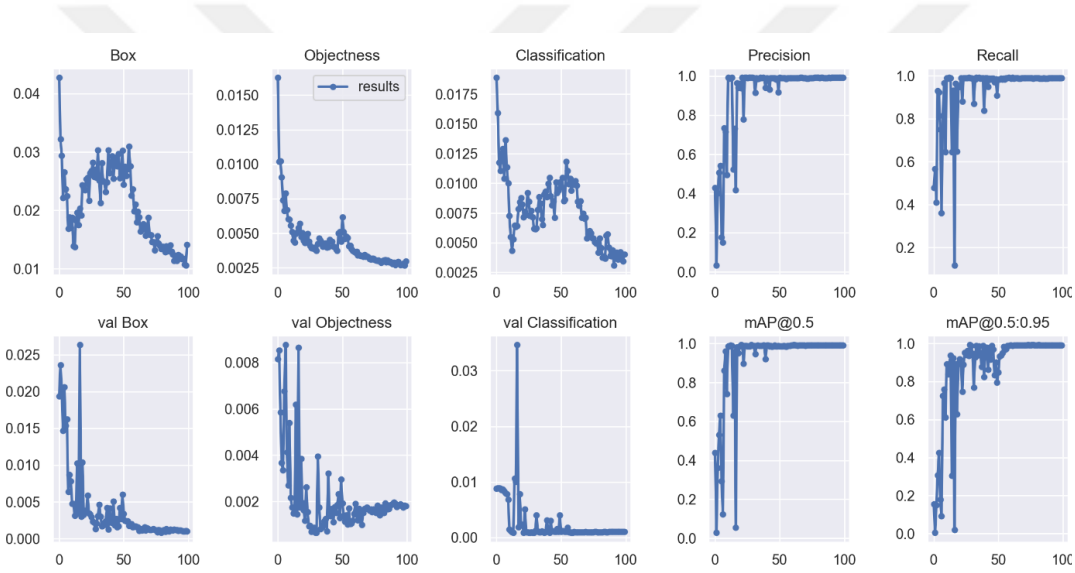
Tablo 4.2. Model eğitim sürecinin performans analizi

Epoch Aralığı	Precision	Recall	mAP@0.5	mAP@0.5:0.95
1-10	0.8	0.75	0.78	0.6
11-20	0.87	0.82	0.84	0.68
21-30	0.91	0.87	0.89	0.73
31-40	0.94	0.9	0.92	0.79
41-50	0.96	0.92	0.94	0.83
51-60	0.97	0.94	0.96	0.88
61-70	0.98	0.95	0.97	0.9
71-80	0.985	0.96	0.975	0.92
81-90	0.99	0.98	0.98	0.94
91-100	**0.995**	**0.99**	**0.985**	**0.96**

4.1.1.6. Eğitim Sürecine Ait Kayıp ve Performans Metriklerinin Görselleştirilmesi

Şekil 4.3`te grafiklerinde eğitim sürecine ait çeşitli performans metriklerinin epoch'lara göre değişimi detaylı şekilde sunulmuştur. Grafiklerin sol kısmında yer alan Box, Objectness ve Classification loss eğrileri, modelin hata oranlarının eğitim süresince belirgin biçimde azaldığını göstermektedir. Aynı şekilde, doğrulama (validation) kayıplarının da benzer bir düşüş eğilimi sergilediği görülmektedir. Bu durum, modelin hem eğitim verisine hem de doğrulama verisine karşı iyi bir genelleme yapabildiğini işaret etmektedir. Özellikle sağ taraftaki Precision, Recall, mAP@0.5 ve

mAP@0.5:0.95 grafiklerinde ise modelin başarı oranlarının eğitim süreci boyunca tutarlı bir şekilde arttığı gözlemlenmektedir. Precision ve Recall değerlerinin erken epoch'larda dahi yüksek olması, modelin sınıflandırma başarısının başlangıçtan itibaren güçlü olduğunu ortaya koymaktadır. Eğitim ilerledikçe bu metriklerdeki istikrarın artması, modelin öğrenme sürecini başarılı bir şekilde tamamladığını göstermektedir. mAP (mean Average Precision) değerlerinde görülen artış ise, modelin farklı eşik değerlerinde dahi kararlı ve yüksek doğrulukla tahmin yapabildiğini desteklemektedir. Tüm bu grafiksel veriler ışığında, elde edilen sonuçlar modelin eğitim sürecinin sağlıklı ve verimli geçtiğini açıkça göstermektedir. Özellikle son epoch'larda ulaşılan yüksek metrik değerleri, modelin artık yeterince öğrenmiş ve dengeli hale gelmiş olduğunu ortaya koymaktadır.



Şekil 4.4. Eğitim Süreci Analiz Grafiği

4.1.1.7. Eğitim Sürecinin Genel Değerlendirmesi

Eğitim süreci boyunca kayıp fonksiyonlarının incelenmesi, modelin her epoch ilerledikçe öğrenme sürecini geliştirdiğini ve hata oranlarını azalttığını göstermektedir. Box loss, objectness loss ve classification loss değerlerinin düşmesi, modelin nesne tespiti, varlık belirleme ve sınıflandırma görevlerinde giderek daha doğru hale geldiğini kanıtlamaktadır. Eğitim sürecinin sonunda mAP@0.5 metriğinin %99 seviyesine ulaşması, modelin genel başarı oranının oldukça yüksek olduğunu desteklemektedir. Son olarak, modelin performansını artırmak için farklı veri artırma teknikleri kullanılmış ve eğitim süreci boyunca çeşitli optimizasyonlar

gerçekleştirilmiştir. Modelin son aşamada yüksek doğruluk oranları elde etmesi, endüstriyel uygulamalarda kullanılabilirliği açısından büyük bir avantaj sağlamaktadır. Bu sonuçlar, modelin gerçek dünya uygulamalarında, özellikle kalite kontrol ve denetim sistemlerinde güvenilir bir şekilde kullanılabileceğini göstermektedir.

4.1.2. Gerçek zamanlı üretim ortamında model performansının değerlendirilmesi

Geliştirilen yapay zeka modeli yalnızca eğitim verisiyle değil, gerçek zamanlı üretim sahasında test edilerek değerlendirilmiştir. Modelin sınıflandırma başarımını detaylı şekilde analiz edebilmek ve sistemin endüstriyel koşullara uyumunu gözlemleyebilmek amacıyla farklı metrikler kullanılarak performans ölçümleri yapılmıştır. Ayrıca sistemin kararlılığını ve doğruluğunu artırmak adına çeşitli görüntü işleme teknikleri uygulanmış, bu yöntemlerin etkileri sistematik olarak incelenmiştir. Elde edilen veriler, modelin üretim ortamında nasıl davrandığını ve hangi optimizasyonların en yüksek başarıyı sağladığını ortaya koymuştur.

4.1.2.1. Görüntü İşleme Tekniklerinin Model Performansına Etkisi

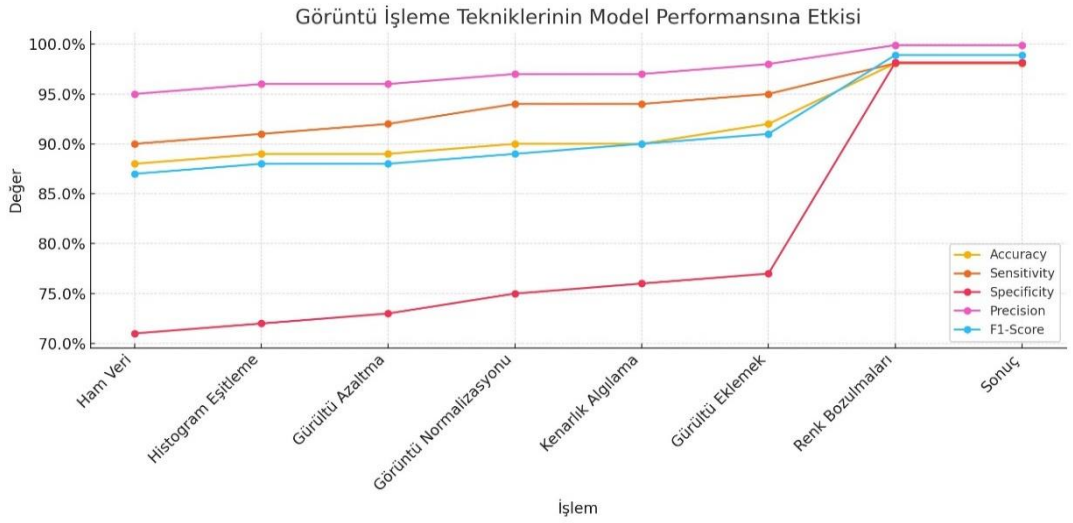
Geliştirilen yapay zeka modelinin üretim ortamındaki başarımını artırmak amacıyla çeşitli görüntü işleme teknikleri uygulanmıştır. Bu bölümde, her bir işlem adımının modelin sınıflandırma performansı üzerindeki etkisi istatistiksel metrikler aracılığıyla değerlendirilmiştir. Başta histogram eşitleme, gürültü azaltma, görüntü normalizasyonu ve kenar algılama gibi geleneksel tekniklerin yanı sıra, veri artırmaya yönelik gürültü ekleme ve renk bozulmaları (color jittering) işlemleri de uygulanmıştır. Yapılan karşılaştırmalar, renk bozulmaları yönteminin modelin doğruluğunu ve genel başarımını anlamlı şekilde artırdığını göstermiştir. İlgili sonuçlar, hem tablo hem de grafik yardımıyla sunularak, görsel karşılaştırmalarla desteklenmiştir. Modelin genel başarımını kapsamlı değerlendirebilmek amacıyla doğruluk (accuracy), duyarlılık (recall), özgüllük (specificity), kesinlik (precision) ve F1-skoru gibi istatistiksel metrikler kullanılmıştır. Bu metrikler aracılığıyla modelin yalnızca doğru ve yanlış sınıflandırma sayılarına değil, aynı zamanda hatalı tahmin türlerine karşı ne kadar dayanıklı olduğu da incelenmiştir. Ayrıca modelin performansını artırmak amacıyla çeşitli görüntü işleme teknikleri uygulanmış ve bu işlemlerin etkisi detaylı olarak değerlendirilmiştir. Özellikle renk bozulmaları (color jittering) gibi veri artırma yöntemlerinin, modelin doğruluk oranını anlamlı biçimde

yükselttiği gözlemlenmiştir. Tablo 4.1`de her bir görüntü işleme tekniğinin model performansı üzerindeki etkisini karşılaştırmalı olarak göstermektedir.

Tablo 4.3. Görüntü işleme sonrası modelin performansları

İşlem	TP	FP	TN	FN	Accuracy	Sensitivity	Specificity	Precision	F1-Score
Ham Veri	900	20	50	30	0.88	0.90	0.71	0.95	0.87
Histogram Eşitleme	902	19	51	28	0.89	0.91	0.72	0.96	0.88
Gürültü Azaltma	904	18	51	27	0.89	0.92	0.73	0.96	0.88
Görüntü Normalizasyon	906	17	52	25	0.90	0.94	0.75	0.97	0.89
Kenarlık Algılama	908	16	52	24	0.90	0.94	0.76	0.97	0.90
Gürültü Eklemek	910	10	53	22	0.92	0.95	0.77	0.98	0.91
Renk Bozulmaları	914	3	53	19	0.9807	0.9807	0.9815	0.9989	0.989
Sonuç	915	1	53	18	**0.9807**	**0.9807**	**0.9815**	**0.9989**	**0.989**

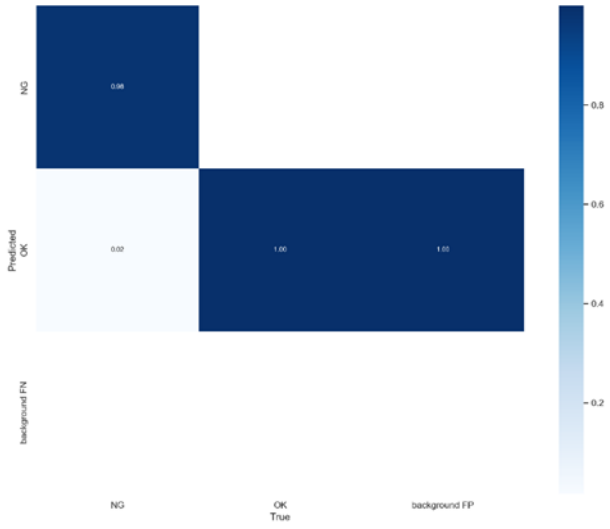
Şekil 4.2`de modelin farklı görüntü işleme teknikleriyle eğitilmesi sonucunda elde edilen istatistiksel performans metriklerini karşılaştırmalı olarak göstermektedir. Bu tekniklerin her biri, modelin doğruluk (accuracy), duyarlılık (sensitivity), özgüllük (specificity), kesinlik (precision) ve F1-ölçümü (F1-score) üzerindeki etkisini görselleştirmek amacıyla analiz edilmiştir. Grafikte görüldüğü üzere, her bir görüntü işleme adımı modelin performansında belirli düzeyde iyileşmeler sağlamıştır. Özellikle renk bozulmaları (color jittering) işlemi, tüm metriklerde en yüksek değerlere ulaşılarak modelin genel doğruluğunu ve güvenilirliğini en fazla artıran yöntem olmuştur. Bu durum, veri çeşitliliğinin artırılmasının modelin genelleme kabiliyeti üzerindeki pozitif etkisini ortaya koymaktadır.



Şekil 4.4. İşlem Adımlarının Model Üzerindeki Etkisi

4.1.2.2. Confusion matrix analizi

Görüntü işleme tekniklerinin uygulanmasının ardından, modelin sınıflandırma başarımını daha net değerlendirebilmek amacıyla Confusion Matrix (Karışıklık Matrisi) oluşturulmuştur. Bu matris, modelin doğru ve yanlış sınıflandırmalarını dört temel kategoriye ayırarak (TP, FP, TN, FN), tahmin doğruluğunu görsel olarak ortaya koymaktadır. Şekil 4.1'de sunulan confusion matrix, tüm iyileştirme adımlarının ardından modelin ulaştığı son sınıflandırma başarımını özetlemektedir. Matriste görüldüğü üzere, NG (hatalı) sınıfında %98, OK (doğru) sınıfında ise %100 doğruluk oranı elde edilmiştir. Yanlış negatif oranı (False Negative - FN) %2 düzeyinde kalmış, yanlış pozitif oranı (False Positive - FP) ise sıfır olarak gerçekleşmiştir.



Şekil 4.5. Confusion Matris

Confusion matrix'e göre**,** model hatalı ürünleri büyük oranda doğru şekilde tespit etmiş ve doğru ürünler için neredeyse hiç hata yapmamıştır. NG (hatalı) sınıfında %98 doğruluk oranı elde edilmesi, modelin üretim hattında kritik öneme sahip olan kusurlu parçaları yüksek doğrulukla ayırt edebildiğini göstermektedir. OK (Dogru) sınıfında %100 doğruluk oranına ulaşılması ise modelin doğru parçaları hatalı olarak işaretleme gibi yanlış pozitif (FP) durumlarının olmadığını ortaya koymaktadır. Bu durum, modelin yanlış alarmlardan kaçınarak operatör müdahalesi gerektiren durumları en aza indirdiğini ifade etmektedir. Yanlış negatif (FN) oranının %2 seviyesinde kalması da modelin hatalı ürünleri gözden kaçırma olasılığının oldukça düşük olduğunu göstermektedir.

4.1.3. Modelin sahada performans analizi

Tüm bu değerlendirmeler doğrultusunda geliştirilen YOLOv7-tiny tabanlı yapay zeka modeli, üretim hattı koşullarında başarıyla uygulanmış ve gerçek zamanlı kalite kontrol süreçlerinde güvenilir sonuçlar vermiştir. Modelin yüksek doğruluk oranı, düşük yanlış negatif ve sıfıra yakın yanlış pozitif oranları sayesinde hem hatalı ürünlerin etkin biçimde tespit edilmesi sağlanmış hem de sağlam ürünlerin gereksiz yere elenmesi önlenmiştir. Görüntü işleme tekniklerinin katkısıyla performansı iyileştirilen bu sistem, sahada yapılan testlerde kararlı ve hızlı çalışarak endüstriyel süreçlerle tam uyumlu bir yapı sergilemiştir.

5. TARTIŞMA

Bu çalışmada, YOLOv7-tiny algoritması kullanılarak endüstriyel nesne tanıma üzerine odaklanılmış ve geliştirilen modelin performansı detaylı bir şekilde analiz edilmiştir. Elde edilen sonuçlar, modelin yüksek doğruluk oranı ve hızlı çalışma süresi ile gerçek zamanlı uygulamalar için uygun olduğunu göstermektedir. Benzer çalışmalarla karşılaştırıldığında, YOLOv7-tiny modelinin doğruluk oranı %98.07 seviyesine ulaşmıştır. Beyin tümörü tanısı üzerine yapılan benzer bir çalışmada YOLOv7 algoritması tabanlı bir karar destek sisteminin doğruluk oranı %97 olarak hesaplanmıştır [17]. Buna karşılık, Faster R-CNN, SSD ve YOLOv3 gibi yaygın kullanılan modellerin doğruluk oranları sırasıyla %85, %74 ve %80 olarak hesaplanmıştır [18]. Bu sonuçlar, YOLOv7-tiny modelinin endüstriyel ve tıbbi görüntü analizi gibi farklı alanlarda yüksek doğruluk sağlayabildiğini göstermektedir. Modelin en büyük avantajlarından biri de hız performansıdır. YOLOv7-tiny, 160 FPS ile diğer derin öğrenme tabanlı nesne tespit algoritmalarına kıyasla çok daha yüksek bir hız sunmuştur. Örneğin, Faster R-CNN 8 FPS, SSD 46 FPS ve YOLOv3 25 FPS değerlerinde kalmıştır [18]. Bu karşılaştırmalar, YOLOv7-tiny modelinin yalnızca yüksek doğruluk sağlamakla kalmayıp, aynı zamanda gerçek zamanlı uygulamalar için en uygun modellerden biri olduğunu göstermektedir. Geliştirilen modelin başarısını artırmak için çeşitli görüntü işleme teknikleri uygulanmıştır. Görüntü normalizasyonu, histogram eşitleme, gürültü azaltma (Gaussian ve Median Blur), kenar algılama, gürültü ekleme (Gaussian ve Salt-Pepper), renk bozulmaları (color jittering) gibi yöntemler sayesinde modelin yanlış pozitif ve yanlış negatif oranları azaltılmıştır. Özellikle, renk bozulmalarının (color jittering) modelin doğruluk oranını önemli ölçüde artırdığı gözlemlenmiştir. Modelin performans analizleri doğrultusunda, color jittering tekniği uygulandığında modelin doğruluk oranının belirgin şekilde arttığı ve sınıflandırma başarımının iyileştiği gözlemlenmiştir [17]. Elde edilen sonuçlar, YOLOv7-tiny modelinin endüstriyel nesne tespiti ve kalite kontrol süreçleri gibi alanlarda başarıyla kullanılabileceğini göstermektedir. Modelin hem yüksek doğruluk oranı hem de hızlı işlem kapasitesi, otomotiv, üretim hatları ve tıbbi görüntüleme gibi

farklı sektörlerde gerçek zamanlı uygulamalar için ideal bir çözüm sunduğunu ortaya koymaktadır.

6. SONUÇ

Bu çalışma, YOLOv7-tiny modelini kullanarak endüstriyel nesne tespiti üzerine odaklanmış ve geliştirilen sistemin performansını çeşitli metriklerle detaylı bir şekilde analiz etmiştir. Yapılan değerlendirmeler sonucunda modelin hem yüksek doğruluk oranı hem de hızlı çalışma kapasitesi ile gerçek zamanlı uygulamalar için uygun bir çözüm sunduğu ortaya konmuştur. Modelin doğruluğu %98.07 olarak hesaplanmış olup, bu oran endüstriyel kalite kontrol süreçleri ve nesne tespiti uygulamalarında yüksek güvenilirlik sağladığını göstermektedir. Elde edilen bu doğruluk seviyesi, Faster R-CNN (%85), SSD (%74) ve YOLOv3 (%80) gibi yaygın kullanılan diğer modellerle karşılaştırıldığında daha üstün bir performans sergilediğini ortaya koymaktadır [18]. Bunun yanı sıra, YOLOv7-tiny'nin 160 FPS hızına ulaşması, modelin gerçek zamanlı uygulamalarda diğer derin öğrenme tabanlı nesne tespit algoritmalarına kıyasla çok daha verimli çalıştığını kanıtlamaktadır [18]. Faster R-CNN, SSD ve YOLOv3 modellerinin sırasıyla 8 FPS, 46 FPS ve 25 FPS seviyelerinde kaldığı göz önüne alındığında, YOLOv7-tiny'nin gerçek zamanlı sistemlerde büyük bir avantaj sağladığı görülmektedir. Bu çalışmada geliştirilen sistem, endüstriyel üretim hatlarında, insan hatalarını minimize ederek gerçek zamanlı nesne tespiti yapabilen bir yapay zeka modeli sunmaktadır. Modelin bu başarısı, uygulanan çeşitli görüntü işleme teknikleri sayesinde elde edilmiştir. Görüntü normalizasyonu, histogram eşitleme, gürültü azaltma (Gaussian ve Median Blur), kenar algılama, gürültü ekleme (Gaussian ve Salt-Pepper) ve renk bozulmaları (color jittering) gibi yöntemler kullanılarak modelin tespit başarımı iyileştirilmiştir. Özellikle renk bozulmalarının (color jittering) modelin doğruluk oranını önemli ölçüde artırdığı gözlemlenmiş olup, bu veri artırma yöntemi sayesinde modelin genelleme yeteneği güçlendirilmiştir [17]. Yapılan analizler, modelin hem duyarlılık (recall) hem de kesinlik (precision) açısından oldukça dengeli bir performans sergilediğini göstermektedir. F1-score metriği 0.989 seviyesinde olup, modelin hem yanlış pozitif hem de yanlış negatif tahminleri düşük tuttuğunu kanıtlamaktadır. Modelin yüksek duyarlılığı, gerçek pozitifleri tespit etme başarısını ortaya koyarken, yüksek kesinlik oranı ise yanlış pozitiflerin minimum seviyede olduğunu göstermektedir. Bunun yanı

sıra, özgüllük (specificity) metriği de oldukça yüksek bir seviyede çıkmış olup, modelin negatif sınıfları başarılı bir şekilde ayırt edebildiğini göstermektedir. Sonuç olarak, bu çalışma YOLOv7-tiny modelinin endüstriyel nesne tespiti ve kalite kontrol süreçlerinde etkin bir şekilde kullanılabileceğini kanıtlamıştır. Modelin yüksek doğruluk oranı, düşük hata payı ve hızlı çalışma süresi, üretim hatları ve diğer otomasyon sistemleri için güçlü bir çözüm sunduğunu göstermektedir. YOLOv7-tiny'nin modern mimarisi ve optimize edilmiş özellikleri, özellikle gerçek zamanlı uygulamalarda hız ve doğruluk açısından büyük bir avantaj sağladığını ortaya koymaktadır. Bu bağlamda, gelecekte yapılacak çalışmalar, modelin daha geniş veri setleriyle eğitilerek farklı endüstriyel uygulamalarda kullanımını test edebilir ve performansını daha da geliştirebilir.

KAYNAKLAR

[1] K. Xu, N. Ragot, and Y. Dupuis, "View Selection for Industrial Object Recognition," in *IECON Proceedings (Industrial Electronics Conference)*, IEEE Computer Society, 2022. doi: 10.1109/IECON49645.2022.9969121.

[2] A. Shrestha, N. Karki, R. Yonjan, M. Subedi, and S. Phuyal, "Automatic Object Detection and Separation for Industrial Process Automation," in *2020 IEEE International Students' Conference on Electrical, Electronics and Computer Science, SCEECS 2020*, Institute of Electrical and Electronics Engineers Inc., Feb. 2020. doi: 10.1109/SCEECS48394.2020.195.

[3] C. Boukouvalas *et al.*, "ASSIST: automatic system for surface inspection and sorting of tiles," 1998.

[4] H. M. T. Abbas, U. Shakoor, M. J. Khan, M. Ahmed, and K. Khurshid, "Automated sorting and grading of agricultural products based on image processing," in *2019 8th International Conference on Information and Communication Technologies, ICICT 2019*, 2019. doi: 10.1109/ICICT47744.2019.9001971.

[5] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You only look once: Unified, real-time object detection," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2016. doi: 10.1109/CVPR.2016.91.

[6] 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, 2017, doi: 10.1109/TPAMI.2016.2577031.

[7] W. Liu *et al.*, "SSD: Single shot multibox detector," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2016. doi: 10.1007/978-3-319-46448-0_2.

[8] J. Redmon and A. Farhadi, "YOLO9000: Better, faster, stronger," in *Proceedings - 30th IEEE Conference on Computer Vision and Pattern Recognition, CVPR 2017*, 2017. doi: 10.1109/CVPR.2017.690.

[9] R. Girshick, J. Donahue, T. Darrell, and J. Malik, "Rich feature hierarchies for accurate object detection and semantic segmentation," in *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 2014. doi: 10.1109/CVPR.2014.81.

[10] O. Ronneberger, P. Fischer, and T. Brox, "U-net: Convolutional networks for biomedical image segmentation," in *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, 2015. doi: 10.1007/978-3-319-24574-4_28.

[11] L. C. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "DeepLab: Semantic Image Segmentation with Deep Convolutional Nets, Atrous Convolution, and Fully Connected CRFs," *IEEE Trans Pattern Anal Mach Intell*, vol. 40, no. 4, 2018, doi: 10.1109/TPAMI.2017.2699184.

[12] X. Glorot, A. Bordes, and Y. Bengio, "Domain adaptation for large-scale sentiment classification: A deep learning approach," in *Proceedings of the 28th International Conference on Machine Learning, ICML 2011*, 2011.

[13] A. Raghunathan, S. M. Xie, F. Yang, J. C. Duchi, and P. Liang, "Understanding and mitigating the tradeoff between robustness and accuracy," in *37th International Conference on Machine Learning, ICML 2020*, 2020.

[14] E. Shelhamer, J. Long, and T. Darrell, "Fully Convolutional Networks for Semantic Segmentation," *IEEE Trans Pattern Anal Mach Intell*, vol. 39, no. 4, 2017, doi: 10.1109/TPAMI.2016.2572683.

[15] L. Binyan, W. Yanbo, C. Zhihong, L. Jiayu, and L. Junqin, "Object detection and robotic sorting system in complex industrial environment," in *Proceedings - 2017 Chinese Automation Congress, CAC 2017*, 2017. doi: 10.1109/CAC.2017.8244092.

[16] Y. Zuo, J. Wang, and J. Song, "Application of YOLO Object Detection Network in Weld Surface Defect Detection," in *2021 IEEE 11th Annual International Conference on CYBER Technology in Automation, Control, and Intelligent Systems, CYBER 2021*, Institute of Electrical and Electronics Engineers Inc., Jul. 2021, pp. 704–710. doi: 10.1109/CYBER53097.2021.9588269.

[17] S. YILMAZ, "Beyin Tümörü Tanıları İçin YOLOv7 Algoritması Tabanlı Karar Destek Sistemi Tasarımı," *Kocaeli Üniversitesi Fen Bilimleri Dergisi*, vol. 6, no. 1, pp. 47–56, Jul. 2023, doi: 10.53410/koufbd.1236305.

[18] S. Srivastava, A. V. Divekar, C. Anilkumar, I. Naik, V. Kulkarni, and V. Pattabiraman, "Comparative analysis of deep learning image detection algorithms," *J Big Data*, vol. 8, no. 1, p. 66, Dec. 2021, doi: 10.1186/s40537-021-00434-w.

EKLER

EK A.Veri Toplama Kodu

```
from threading import Thread, Lock
from pypylon import pylon
import cv2
import datetime
import logging
import os
import sys
import time

class CameraThread(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.date = datetime.datetime.now()

        # Logger
        self.logger = logging.getLogger("Camera")
        self.logger.setLevel(logging.INFO)
        file_handler = logging.FileHandler("app_basler_camera.log")
        file_handler.setLevel(logging.INFO)
        formatter = logging.Formatter(fmt="%(asctime)s - %(name)s - %(levelname)s
- %(message)s",
                                      datefmt="%d/%m/%Y %H:%M:%S")
        file_handler.setFormatter(formatter)
        self.logger.addHandler(file_handler)
        self.logger.addHandler(logging.StreamHandler(sys.stdout))
        self.logger.warning("Initial log. Camera started!")

        self.PATH = os.path.dirname(os.path.abspath(__file__))
        self.isCameraConnected = False
        self.frame = None
        self.lock = Lock()
        self.is_running = True  # Kamera thread'ini kontrol etmek için değişken

        try:
            self.camera =
pylon.InstantCamera(pylon.TlFactory.GetInstance().CreateFirstDevice())
            self.camera.StartGrabbing(pylon.GrabStrategy_LatestImageOnly)
            self.converter = pylon.ImageFormatConverter()
            self.converter.OutputPixelFormat = pylon.PixelType_BGR8packed
            self.converter.OutputBitAlignment = pylon.OutputBitAlignment_MsbAligned
            self.isCameraConnected = True
            self.logger.info("Basler Camera successfully initialized.")
        except Exception as e:
            self.logger.exception(f"Error initializing camera: {e}")
            raise Exception("Camera initialization failed!")

    def run(self):
        """ Kamera görüntüsünü alır ve sürekli günceller. """
        try:
            while self.camera.IsGrabbing() and self.is_running:
                grabResult = self.camera.RetrieveResult(5000,
pylon.TimeoutHandling_ThrowException)
                if grabResult.GrabSucceeded():
                    image = self.converter.Convert(grabResult)
                    with self.lock:
                        self.frame = image.GetArray()

                    self.logger.info("Image successfully grabbed from Basler
camera.")
                grabResult.Release()
        except Exception as e:
            self.logger.exception(f"Camera connection lost: {e}")
```

```
            self.isCameraConnected = False
            raise Exception("Camera connection lost!")

    def show_frame(self):
        """ Kamera görüntüsünü ekranda gösterir. """
        while self.is_running:
            with self.lock:
                if self.frame is not None and self.frame.size > 0:
                    cv2.imshow("Camera Feed", self.frame)

            if cv2.waitKey(1) & 0xFF == ord('q'):
                self.stop()  # 'q' tuşuna basılınca çıkış yap
                break

        cv2.destroyAllWindows()

    def get_frame(self):
        """ Mevcut frame'i döndürür. """
        with self.lock:
            return self.frame.copy() if self.frame is not None else None

    def img_store(self, pic_path):
        """ Kameradan alınan görüntüyü sürekli kaydetmek için ayrı bir thread. """
        if not self.isCameraConnected:
            self.logger.error("Cannot save image: Camera is not connected")
            return

        while self.is_running:
            date = datetime.datetime.now()
            str_date = date.strftime("%Y-%m-%d-%H%M%S%f")
            img_path = os.path.join(self.PATH, pic_path, f"{str_date}.png")

            with self.lock:
                if self.frame is not None and self.frame.size > 0:
                    cv2.imwrite(img_path, self.frame)
                    self.logger.info(f"Image saved at: {img_path}")
                else:
                    self.logger.error("Image not saved: Frame is empty")

            time.sleep(1)

    def stop(self):
        """ Kamera ve diğer işlemleri güvenli şekilde durdurur. """
        self.is_running = False
        self.logger.info("Stopping all camera processes...")

        # Kameranın grab işlemini durdur
        if self.camera.IsGrabbing():
            self.camera.StopGrabbing()

        # OpenCV pencerelerini kapat
        cv2.destroyAllWindows()

        self.logger.info("All processes stopped successfully.")
```

```
from camera import CameraThread
import threading
import cv2

# Kamera başlat
cam = CameraThread()
cam.start()  # Kamera thread'ini başlat
# Görüntüyü gösterme thread'i başlat
show_thread = threading.Thread(target=cam.show_frame)
show_thread.start()
# Görüntü kaydetme işlemini ayrı bir thread olarak başlat
store_thread = threading.Thread(target=cam.img_store,
args=(r"C:\Users\MT33213\Desktop\SwProject\04.AI_Cam\deneme",))
store_thread.start()

# Ana thread: "q" tuşuna basılınca tüm işlemleri durdur
while cam.is_running:
    if cv2.waitKey(1) & 0xFF == ord('q'):  # "q" tuşuna basılınca çık
        print("Durduruluyor...")
        cam.stop()  # Kamera thread'ini durdur
        break
```

```
# Thread'leri güvenli şekilde kapat
cam.join()
show_thread.join()
store_thread.join()
print("Tüm işlemler durduruldu.")
```

EK B. Veri Çoğaltma Kodu

```
def process_and_save_images(input_folder, output_folder, processing_functions):
    """
    Belirtilen klasördeki tüm görüntüleri işleyip sonuçları kaydeder.

    :param input_folder: İşlenecek görüntülerin bulunduğu klasör
    :param output_folder: İşlenmiş görüntülerin kaydedileceği klasör
    :param processing_functions: Kullanılacak görüntü işleme fonksiyonlarının listesi
    """
    if not os.path.exists(output_folder):
        os.makedirs(output_folder)  # Çıkış klasörü yoksa oluştur

    # Klasördeki tüm görüntüleri al
    image_files = [f for f in os.listdir(input_folder) if f.endswith(('.jpg', '.png',
'.jpeg'))]

    for image_file in image_files:
        image_path = os.path.join(input_folder, image_file)
        image = cv2.imread(image_path)

        if image is None:
            print(f"Hata: {image_file} yüklenemedi. Dosya bozuk olabilir.")
            continue

        print(f"İşleniyor: {image_file}")

        for func in processing_functions:
            processed_image = func(image)  # Fonksiyonu uygula
            processed image name =
f"{os.path.splitext(image_file)[0]}_{func.__name__}.png"
            processed_image_path = os.path.join(output_folder, processed_image_name)

            cv2.imwrite(processed_image_path, processed_image)  # İşlenmiş görüntüyü
kaydet
            print(f"Kaydedildi: {processed image path}")

    print(" Tüm görüntüler işlendi ve kaydedildi!")

#  Görüntü işleme fonksiyonlarını düzenleyerek, çıktıyı döndürmelerini sağlayalım
def apply_histogram_equalization(image):
    """ Histogram eşitleme uygular ve sonucu döndürür. """
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    return cv2.equalizeHist(gray)

def convert_color_space(image):
    """ Görüntüyü HSV'ye dönüştürüp geri döndürür. """
    return cv2.cvtColor(image, cv2.COLOR_BGR2HSV)

def normalize_image(image, method="min-max"):
    """ Görüntüyü normalize eder ve sonucu döndürür. """
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY).astype(np.float32)
    if method == "min-max":
        return (gray / 255.0 * 255).astype(np.uint8)
    elif method == "z-score":
        mean, std = np.mean(gray), np.std(gray)
        return ((gray - mean) / (std + 1e-7) * 255).astype(np.uint8)
    return gray

def apply_noise_reduction(image, blur_type="gaussian", kernel_size=5):
    """ Gürültü azaltma (Gaussian veya Median Blur) uygular ve sonucu döndürür. """
    if blur_type == "gaussian":
        return cv2.GaussianBlur(image, (kernel_size, kernel_size), 0)
    elif blur_type == "median":
        return cv2.medianBlur(image, kernel_size)
    return image

def apply_edge_detection(image, method="sobel", threshold1=100, threshold2=200,
ksize=3):
    """ Kenar algılama uygular ve sonucu döndürür. """
    gray = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)
    if method == "canny":
        return cv2.Canny(gray, threshold1, threshold2)
    elif method == "sobel":
        sobelx = cv2.Sobel(gray, cv2.CV_64F, 1, 0, ksize=ksize)
```

```
        sobely = cv2.Sobel(gray, cv2.CV_64F, 0, 1, ksize=ksize)
        return cv2.convertScaleAbs(cv2.magnitude(sobelx, sobely))
    elif method == "laplacian":
        return cv2.convertScaleAbs(cv2.Laplacian(gray, cv2.CV 64F, ksize=ksize))
    return gray

def add_noise(image, noise_type="gaussian", amount=0.02, mean=0, std=25):
    """ Gaussian veya Tuz & Biber gürültüsü ekler ve sonucu döndürür. """
    noisy image = np.copy(image)

    if noise_type == "gaussian":
        noise = np.random.normal(mean, std, image.shape).astype(np.float32)
        noisy_image = cv2.add(image.astype(np.float32), noise)
        return np.clip(noisy_image, 0, 255).astype(np.uint8)

    elif noise_type == "salt_pepper":
        row, col, ch = image.shape
        num_salt = np.ceil(amount * row * col)
        num_pepper = np.ceil(amount * row * col)

        coords = [np.random.randint(0, i, int(num_salt)) for i in image.shape[:2]]
        noisy_image[coords[0], coords[1], :] = 255

        coords = [np.random.randint(0, i, int(num_pepper)) for i in image.shape[:2]]
        noisy_image[coords[0], coords[1], :] = 0

        return noisy_image
    return image

def apply_color_jitter(image, brightness=0.2, contrast=0.2, saturation=0.2, hue=0.1):
    """ Renk bozulmaları (Color Jittering) uygular ve sonucu döndürür. """
    brightness_factor = 1 + np.random.uniform(-brightness, brightness)
    contrast_factor = 1 + np.random.uniform(-contrast, contrast)
    jittered_image = cv2.convertScaleAbs(image, alpha=contrast_factor,
beta=brightness_factor * 50)

    hsv_image = cv2.cvtColor(jittered_image, cv2.COLOR_BGR2HSV).astype(np.float32)

    saturation_factor = 1 + np.random.uniform(-saturation, saturation)
    hsv_image[:, :, 1] = np.clip(hsv_image[:, :, 1] * saturation_factor, 0, 255)

    hue_factor = np.random.uniform(-hue * 180, hue * 180)
    hsv_image[:, :, 0] = np.clip(hsv_image[:, :, 0] + hue_factor, 0, 179)

    return cv2.cvtColor(hsv_image.astype(np.uint8), cv2.COLOR_HSV2BGR)
```

EK C. Veri Setinin Eğitim ve Doğrulama Olarak Ayırma Kodu

```
def split_dataset_copy(source_folder, train_ratio=0.7):
    """
    .png ve .txt dosyalarını belirtilen orana göre training ve validation
klasörlerine KOPYALAR.

    :param source_folder: .png ve .txt dosyalarının bulunduğu ana klasör (str)
    :param train_ratio: Eğitim seti oranı (0-1 arası float, örn: 0.7 → %70
eğitim, %30 doğrulama)
    """
    # Hedef klasörler
    train_data_dir = os.path.join(source_folder, "training/data")
    train_labels_dir = os.path.join(source_folder, "training/labels")
    val_data_dir = os.path.join(source_folder, "validation/data")
    val_labels_dir = os.path.join(source_folder, "validation/labels")

    # Klasörleri oluştur
    for folder in [train_data_dir, train_labels_dir, val_data_dir, val_labels_dir]:
        os.makedirs(folder, exist_ok=True)

    # Tüm .png dosyalarını al
    images = [f for f in os.listdir(source_folder) if f.endswith(".png")]

    # Karıştır ve böl
    random.shuffle(images)
    train_count = int(len(images) * train_ratio)

    train_images = images[:train_count]
    val_images = images[train_count:]

    # Dosyaları yeni konumlarına KOPYALA
    for image in train_images:
        image_path = os.path.join(source_folder, image)
        label_path = os.path.join(source_folder, os.path.splitext(image)[0] +
".txt")  # İlgili .txt dosyası

        shutil.copy(image_path, os.path.join(train_data_dir, image))  # Görüntüyü
kopyala
        if os.path.exists(label_path):  # Eğer etiket dosyası varsa kopyala
            shutil.copy(label_path, os.path.join(train_labels_dir,
os.path.basename(label_path)))

    for image in val_images:
        image_path = os.path.join(source_folder, image)
        label_path = os.path.join(source_folder, os.path.splitext(image)[0] +
".txt")  # İlgili .txt dosyası

        shutil.copy(image_path, os.path.join(val_data_dir, image))  # Görüntüyü
kopyala
        if os.path.exists(label_path):  # Eğer etiket dosyası varsa kopyala
            shutil.copy(label_path, os.path.join(val_labels_dir,
os.path.basename(label_path)))

    print(f" {train_count} görüntü ve etiket training setine KOPYALANDI.")
    print(f" {len(val_images)} görüntü ve etiket validation setine KOPYALANDI.")

#  Kullanım: Kaynak klasörü belirt ve train-validation oranını ayarla
source_folder =
r"C:\Users\MT33213\Desktop\SwProject\04.AI_Cam\EgitimDatasi\DataCogaltma\Processed_Im
agesOK"  # Kendi veri klasörünü gir
split_dataset_copy(source_folder, train_ratio=0.7)  # %70 eğitim, %30 doğrulama
```

EK D. Python ile Kamera Başlatma ve Görüntü Alımı

```
from threading import Thread, Lock
from pypylon import pylon
import cv2
import datetime
import logging
import os
import sys
import time

class CameraThread(Thread):
    def __init__(self):
        Thread.__init__(self)
        self.date = datetime.datetime.now()

        # Logger
        self.logger = logging.getLogger("Camera")
        self.logger.setLevel(logging.INFO)
        file_handler = logging.FileHandler("app_basler_camera.log", encoding="utf-8")
        file_handler.setLevel(logging.INFO)
        formatter = logging.Formatter(fmt="%(asctime)s - %(name)s - %(levelname)s
- %(message)s",
                                      datefmt="%d/%m/%Y %H:%M:%S")
        file_handler.setFormatter(formatter)
        self.logger.addHandler(file_handler)
        self.logger.addHandler(logging.StreamHandler(sys.stdout))
        self.logger.warning("Initial log. Camera started!")

        self.PATH = os.path.dirname(os.path.abspath(__file__))
        self.frame = None
        self.lock = Lock()
        self.is_running = True  # Kamera thread'ini kontrol etmek için değişken

        try:
            self.camera =
pylon.InstantCamera(pylon.TlFactory.GetInstance().CreateFirstDevice())
            self.camera.StartGrabbing(pylon.GrabStrategy_LatestImageOnly)
            self.converter = pylon.ImageFormatConverter()
            self.converter.OutputPixelFormat = pylon.PixelType_BGR8packed
            self.converter.OutputBitAlignment = pylon.OutputBitAlignment_MsbAligned
            self.logger.info("Basler Camera successfully initialized.")
        except Exception as e:
            self.logger.exception(f"Error initializing camera: {e}")
            raise Exception("Camera initialization failed!")

    def run(self):
        """  Kamera açık kalacak ama görüntü sürekli alınmayacak """
        while self.is_running and self.camera.IsGrabbing():
            time.sleep(0.1)  # CPU kullanımını azaltmak için kısa bekleme süresi

    def get_frame(self):
        """  Kamera açıldığında sadece bu fonksiyon çağrıldığında görüntü alınacak
"""
        if not self.camera.IsGrabbing():
            self.logger.error(" Kamera şu anda görüntü almıyor.")
            return None

        try:
            grabResult = self.camera.RetrieveResult(5000,
pylon.TimeoutHandling_ThrowException)
            if grabResult.GrabSucceeded():
                image = self.converter.Convert(grabResult)
                with self.lock:
                    self.frame = image.GetArray()
                self.logger.info("📸 Tek bir görüntü başarıyla alındı.")
            grabResult.Release()
        except Exception as e:
            self.logger.exception(f" Kamera hatası: {e}")
            return None

        return self.frame

    def stop(self):
        """  Kamera ve thread işlemlerini güvenli şekilde kapatır """
```

```
        self.is_running = False
        if self.camera.IsGrabbing():
            self.camera.StopGrabbing()
        self.logger.info(" Kamera kapatıldı.")
```

EK E. Modbus TCP Kullanılarak PLC Verisi Okuma/Yazma Kodu

```
import socket
import time
from time import sleep
from threading import Thread
from random import uniform
import datetime
import logging
import sys
import os
from pymodbus.client.sync import ModbusTcpClient

class Modbus(Thread):
    def __init__(self,tcp_ip,tcp_port):
        Thread.__init__(self)

        self.TCP_IP = tcp_ip
        self.TCP_PORT = tcp_port
        self.BUFFER_SIZE = 2048
        # Logger
        self.logger = logging.getLogger("PLC")
        self.logger.setLevel(logging.INFO)
        # self.fileHandler = logging.FileHandler("app.log")
        self.fileHandler = logging.FileHandler("app_dummy_plc.log")
        self.fileHandler.setLevel(logging.INFO)
        self.logger.addHandler(self.fileHandler)
        self.formatter = logging.Formatter(fmt="%(asctime)s - %(name)s
- %(levelname)s - %(message)s",
                                           datefmt="%d/%m/%Y %H:%M:%S")
        self.fileHandler.setFormatter(self.formatter)
        self.logger.addHandler(self.fileHandler)
        self.logger.addHandler(logging.StreamHandler(sys.stdout))
        self.logger.warning("Initial log. PLC server started!")

        self.PATH = os.path.dirname(os.path.abspath(  file  ))
        self.conn = None
        self.PLCServer = None
        self.resultCoils = []
        self.resultInputRegisters = []
        self.resultHoldingRegister = []
        self.resultDiscreteInputsRegister = []
        self.resultCoils = []
        try:
            self.client = ModbusTcpClient(self.TCP_IP, self.TCP_PORT)
            print("Bağlandı")

        except Exception as e:
            self.logger.exception(f"Error Code: P001\n{str(e)}")

    def run(self):
        while True:
            continue

    def readCoilsRegister(self, start, count):
        """ Modbus Coils Register'larını oku ve sonucu döndür """
        result = self.client.read_coils(start, count)
        if result.isError():
            return []  # Hata olursa boş liste dön
        return result.bits  # **Sonucu döndür**

            #print(self.resultCoils[0])
            #time.sleep(0.05)
            #print(self.resultCoils)

    def writeCoils(self,bit, value):
        self.client.write_coil(bit, value)

    def readInputRegister(self,start,count):

        self.resulInputRegisters = self.client.read_input_registers(start,count)

    def readHoldingRegister(self,start,count):
        while True:
```

```
            self.resultHoldingRegister =
self.client.read_holding_registers(start,count)
            self.resultHoldingRegister = self.resultHoldingRegister.registers
            time.sleep(1)

    def readDiscreteInputs(self,start,count):
        self.resultDiscreteInputsRegister =
self.client.read_discrete_inputs(start,count)

    def closeConnection(self):
        self.client.close()
```

EK F. YOLOv7 ile NG/OK Tespiti için Görüntü Analizi Kodu

```
import os
import torch
import cv2
import numpy as np
import datetime
import sys
from utils.datasets import letterbox
# **YOLOv7 Klasörünü Python Yoluna Ekle**
CUR_PATH = os.path.dirname(os.path.abspath(__file__))
sys.path.append(os.path.join(CUR_PATH, "yolov7"))

# **YOLOv7'nin Kendi Modüllerini İçe Aktar**
from models.experimental import attempt_load
from utils.general import non_max_suppression, scale_coords
from utils.torch_utils import select_device

import warnings
warnings.filterwarnings("ignore", category=UserWarning)

class AI():
    def __init__(self):
        self.labels = ["NG", "OK"]
        self.colors = [(0, 255, 255), (0, 0, 255)]

        # **Cihazı belirle (CUDA varsa GPU kullan, yoksa CPU)**
        self.device = select_device("cuda" if torch.cuda.is_available() else "cpu")

        # **Modeli yükle**
        model_path = os.path.join(CUR_PATH, "aimodel", "yolov7-tiny.pt")  # Model
dosyanın yolu
        self.model = attempt_load(model_path, map_location=self.device)  # YOLOv7'nin
model yükleyicisi
        self.model.to(self.device).eval()

        #   **Klasörlerin oluşturulması**
        for folder in ["images", "images/NG", "images/OK", "images/NULL"]:
            os.makedirs(os.path.join(CUR_PATH, folder), exist_ok=True)

    def detection(self, userImage):
        DATE = datetime.datetime.now()
        STR_DATE = DATE.strftime("%Y-%m-%d-%H%M%S")

        # Orijinal görüntüyü oku
        imagetoDetect = cv2.imread(userImage)
        if imagetoDetect is None:
            print("  HATA: Görüntü dosyası okunamadı!")
            return 1, 0

        h, w = imagetoDetect.shape[:2]
        target_size = (640, 640)

        # Letterbox yöntemiyle yeniden boyutlandırma (orantıyı korur, dolgu ekler)
        img_letterbox, ratio, pad = letterbox(imagetoDetect, new_shape=target_size,
auto=False)

        # Görüntüyü BGR'den RGB'ye çevirip tensora dönüştürme
        img = cv2.cvtColor(img_letterbox, cv2.COLOR_BGR2RGB)
        img = np.ascontiguousarray(img)
        img = torch.from_numpy(img).permute(2, 0, 1).float().unsqueeze(0) / 255.0
        img = img.to(self.device)

        # Modeli çalıştırma
        with torch.no_grad():
            pred = self.model(img)[0]

        pred = non_max_suppression(pred, conf_thres=0.25, iou_thres=0.45)

        if pred[0] is None or len(pred[0]) == 0:
            print(" UYARI: Hiç nesne tespit edilmedi, yine de görüntü kaydedilecek.")
            cv2.putText(imagetoDetect, "NG - No Objects Detected", (50, 50),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (0, 0, 255), 4)
            cv2.putText(imagetoDetect, STR_DATE, (w - 300, h - 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (135, 143, 145), 4)
```

```
            ng_path = os.path.join(CUR_PATH, "images", "NULL", f"{STR_DATE}.png")
            real_path = os.path.join(CUR_PATH, "images", "real.png")
            cv2.imwrite(ng_path, imagetoDetect)
            cv2.imwrite(real path, imagetoDetect)
            print(f"  AI modeli görüntüyü kaydetti: {real_path}")
            return 1, 0

        best_obj = max(pred[0], key=lambda x: x[4])
        x1, y1, x2, y2, confidence, class id = best obj.cpu().numpy()

        # Letterbox dönüşüm detaylarını kullanarak koordinatları orijinal görüntüye
dönüştürme
        dw, dh = pad  # pad, (dw, dh) şeklinde geliyor
        # ratio, [ratio_width, ratio_height] gibi iki elemanlı bir dizi ise:
        x1 = max(int((x1 - dw) / ratio[0]), 0)
        y1 = max(int((y1 - dh) / ratio[1]), 0)
        x2 = min(int((x2 - dw) / ratio[0]), w - 1)
        y2 = min(int((y2 - dh) / ratio[1]), h - 1)

        class_id = int(class_id)
        confidence = float(confidence)
        label = f"{self.labels[class_id]} {confidence:.2f}"
        print(f"  Tespit Edilen: {label}")

        # Orijinal görüntü üzerinde bounding box çizimi
        box_color = self.colors[class_id]
        cv2.rectangle(imagetoDetect, (x1, y1), (x2, y2), box_color, 4)
        cv2.putText(imagetoDetect, label, (x1, y1 - 10), cv2.FONT_HERSHEY_SIMPLEX, 1,
box_color, 4)

        # Sonuç görüntüsünü kaydetme
        real_path = os.path.join(CUR_PATH, "images", "real.png")
        cv2.imwrite(real_path, imagetoDetect)

        if class_id == 0:
            ng_path = os.path.join(CUR_PATH, "images", "NG", f"{STR_DATE}.png")
            cv2.putText(imagetoDetect, STR_DATE, (imagetoDetect.shape[1] - 300,
imagetoDetect.shape[0] - 30),
                        cv2.FONT_HERSHEY_SIMPLEX, 1, (135, 143, 145), 4)
            cv2.imwrite(ng_path, imagetoDetect)
        else:
            ok_path = os.path.join(CUR_PATH, "images", "OK", f"{STR_DATE}.png")
            smallFile = cv2.resize(imagetoDetect, None, fx=0.5, fy=0.5)
            cv2.imwrite(ok_path, smallFile)

        return class_id, confidence
```

EK G. PyQt5 Tabanlı Kullanıcı Arayüzü Uygulama Kodu

```
import os
from PyQt5.QtWidgets import QApplication, QMainWindow, QWidget, QVBoxLayout,
QHBoxLayout, QLabel, QFrame, QPushButton
from PyQt5.QtGui import QPixmap
from PyQt5.QtCore import Qt
import threading

CUR_PATH = os.path.dirname(os.path.abspath(__file__))

class WorkerThread(threading.Thread):
    def __init__(self):
        super().__init__()

    def run(self):
        # Simulate a background task
        pass

class MyMainWindow(QMainWindow):
    def keyPressEvent(self, event):
        if event.key() == Qt.Key_F11:
            if self.windowState() == Qt.WindowFullScreen:
                self.showNormal()
            else:
                self.showFullScreen()
        elif event.key() == Qt.Key_Escape:
            self.showNormal()
       else:
           super().keyPressEvent(event)

    def __init__(self):
        super().__init__()
        self.setWindowTitle("FSM RR TACK 2 LH AI Camera Control")
        self.screen = QApplication.desktop().screenGeometry()
        self.main_widget = QWidget()
        self.setCentralWidget(self.main_widget)
        self.layout = QVBoxLayout()
        self.main_widget.setLayout(self.layout)
        self.frame = QFrame()
        self.frame.setFrameShape(QFrame.StyledPanel)
        self.frame.setStyleSheet("background-color: black;")
        self.layout.addWidget(self.frame)
        self.frame layout = QVBoxLayout()
        self.frame.setLayout(self.frame_layout)
        self.header_widget = QWidget()
        self.header_widget.setStyleSheet("background-color: rgb(145, 0, 0);")
        self.header_layout = QHBoxLayout()
        self.header widget.setLayout(self.header layout)
        self.header_label = QLabel("FSM RR TACK 2 LH AI Camera Control")
        self.header_label.setStyleSheet("color: white; font-size: 24px ;font-weight:
bold ;font-family: Baskerville ")
        self.header_layout.addWidget(self.header_label)

        self.worker_thread = WorkerThread()
        self.worker_thread.start()

        # Define header_height here
        icon_label = QLabel()
        pixmap = QPixmap("toyota.png")  # Icon resminizin dosya yolu
        self.header_height = int(self.screen.height() * 0.035)  # Header bölgesinin
yüksekliği
        pixmap = pixmap.scaledToHeight(self.header_height - 10,
Qt.SmoothTransformation)
        icon_label.setPixmap(pixmap)
        self.header_layout.addWidget(icon_label, alignment=Qt.AlignRight)

        self.header_widget.setFixedHeight(self.header_height)  # Header bölgesinin
yüksekliğini ayarla
        self.frame_layout.addWidget(
            self.header_widget)  # Header bölgesini çerçeve içindeki düzenleme
yöneticisine ekleyin

        # Add a QLabel for displaying the image
        self.visual_area_label = QLabel()
```

```
        self.current_image = CUR_PATH + r"\images\real.png"
        self.update_image()  # Pass header_height as an argument
        self.frame_layout.addWidget(self.visual_area_label, alignment=Qt.AlignTop)

        # İkinci bağımsız satırı ekleyelim
        self.second_row = QHBoxLayout()

        # İkinci satırın ilk sütunu %70 genişlikte, mavi arka planlı
        first_column = QLabel("Başkabir şey koymak için kullanılabilir")
        first_column.setStyleSheet("background-color: rgb(120, 120, 120);")
        self.second_row.addWidget(first_column, 7, alignment=Qt.AlignTop)

        first_column.setFixedHeight(150)

        # İkinci satırın ikinci sütunu %30 genişlikte, mavi arka planlı

        self.second_column = QLabel("OK")
        self.second_column.setStyleSheet(
            "border-radius: 10px ;qproperty-alignment: AlignCenter;text-align:center;
color: white; font-size: 24px ;font-weight: bold  ;background-color: green;")

        self.second_row.addWidget(self.second_column, 3, alignment=Qt.AlignTop)
        self.second_column.setFixedHeight(150)
        self.frame_layout.addLayout(self.second_row)

        # Title Bölgesi
        visual_area_label2 = QLabel()
        image2 = QPixmap("your_image.png")  # Görsel alanına eklemek istediğiniz
resmin dosya yolu
        image2 = image2.scaled(self.screen.width(), self.screen.height() -
self.header_height, Qt.KeepAspectRatio)
        visual_area_label2.setPixmap(image2)
        self.frame_layout.addWidget(visual_area_label2, alignment=Qt.AlignTop)

    def update_image(self):
        """ QLabel içinde görüntüyü sağdan genişletip tam oturtur """
        if not os.path.exists(self.current_image):
            print(f" UYARI: Görüntü dosyası bulunamadı! {self.current_image}")
            return

       #  **Görüntüyü Yükle**
        image = QPixmap(self.current_image)

        #  **QLabel'in mevcut boyutunu al**
        label_width = self.visual_area_label.width()
        label_height = self.visual_area_label.height()

        #  **Görüntüyü QLabel alanına tam oturtarak ölçekle**
        scaled_image = image.scaled(label_width, label_height, Qt.IgnoreAspectRatio,
Qt.SmoothTransformation)

        #  **Güncellenmiş görüntüyü QLabel'e set et**
        self.visual_area_label.setPixmap(scaled_image)

        print(f" Görüntü güncellendi ve QLabel'e tam oturtuldu:
{self.current_image}")

    def resizeEvent(self, event):
        """ Pencere yeniden boyutlandırıldığında çağrılan fonksiyon """
        super().resizeEvent(event)
        self.update_image()  #  **Görüntüyü yeniden ölçekle**

    def AIresult(self,status):
        if status == 0:
            self.second_column.setText("NG")
            self.second_column.setStyleSheet(
                "border-radius: 10px ;qproperty-alignment: AlignCenter;text-
align:center; color: white; font-size: 24px ;font-weight: bold  ;background-color:
red;"
            )

           print("AI Sonuc NG")
        else:
            self.second_column.setText("OK")
            self.second_column.setStyleSheet(
```

```
                "border-radius: 10px ;qproperty-alignment: AlignCenter;text-align:center;
color: white; font-size: 24px ;font-weight: bold  ;background-color: green;"
            )
            print("AI Sonuc OK")
```

EK H. Kamera, PLC ve Arayüzün Birlikte Çalıştığı Ana Kod

```
import sys
import AImodel
from screenCode import MyMainWindow
from PyQt5.QtWidgets import QApplication
import threading
from modbus import Modbus
from camera import CameraThread
import os
import openpyxl
import datetime
import time
import cv2

# **Global Değişkenler**
CUR_PATH = os.path.dirname(os.path.abspath(__file__))
ImagetoDetectPath = CUR_PATH + r"/images/" + "buffer.png"

workbook_name = "dataAnalysis.xlsx"
workbook_obj = openpyxl.load_workbook(workbook_name)
sheet_obj = workbook_obj.active

# **Programın Çalışmasını Kontrol Eden Değişken**
running = True  #  Program çalışıyorken True olacak

def mainJob(window, modbusCom, camera_thread):
    """ Ana işlem döngüsü """
    global running
    AI = AImodel.AI()  # **AI Modeli sadece bir kez yükleniyor**
    last_trigger_state = 0  # **Son trigger durumu**

    while running:  #  Program kapatılana kadar çalışmaya devam et
        registers = modbusCom.readCoilsRegister(0, 8)  # **Trigger kontrolü**
        current_trigger = registers[0] if registers else 0

        if current_trigger == 1 and last_trigger_state == 0:
            print(" Yeni Trigger Algılandı, Kamera Görüntü Alacak!")

            frame = camera_thread.get_frame()
            if frame is not None:
                img_path = os.path.join(CUR_PATH, "images", "buffer.png")
                cv2.imwrite(img_path, frame)  # **PNG olarak kaydet**

                #  **AI Modeli ile tahmin yap**
                result, conf = AI.detection(img_path)
                print(f"AI Sonucu: {result} - Güven: {conf:.2f}")
                window.AIresult(result)
                window.update_image()

            #  **Sonuçları Excel'e kaydet**
            DATE = datetime.datetime.now()
            current_date = DATE.strftime("%Y-%m-%d-%H%M%S")
            sheet_obj.append([current_date, result, conf])
            workbook_obj.save(workbook_name)

        last_trigger_state = current_trigger  #  Son trigger durumunu güncelle
        time.sleep(0.1)  # **CPU'yu yormamak için bekleme süresi**

def stop_program():
    """ Programı düzgün şekilde durduran fonksiyon """
    global running
    running = False  #  Ana döngüyü durdur
    print("  Program durduruluyor...")

    #  Kamera ve Modbus bağlantısını kapat
    camera_thread.stop()
    modbusCom.closeConnection()

    print(" Tüm işlemler temiz şekilde kapatıldı.")
    sys.exit(0)

if __name__ == "__main__":
    """ Uygulamayı Başlat """
    app = QApplication(sys.argv)
```

```
    window = MyMainWindow()
    window.show()

    #  **Modbus ve Kamera başlatılıyor**
    modbusCom = Modbus('127.0.0.1', 502)
    camera_thread = CameraThread()
    camera_thread.start()

    #  **Ana iş parçacığını (thread) başlat**
    anais_thread = threading.Thread(target=mainJob, args=(window, modbusCom,
camera_thread))
    anais_thread.start()

    #  **"q" tuşuna basıldığında uygulamayı kapat**
    app.aboutToQuit.connect(stop program)  # PyQt5 çıkış sinyali ile programı durdur

    sys.exit(app.exec_())
```

ÖZGEÇMİŞ

Ad-Soyad : Enesalp ÖZ

ÖĞRENİM DURUMU:

- **Lisans :** 2021, Yıldız Teknik Üniversitesi, Elektrik-Elektronik Fakultesi, Elektrik Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- 2020-2021 yılları arasında PNP Elektrik Enerji Otomasyon firmasında stajyer mühendisi olarak çalıştı.
- 2021-2022 yılında Toyota Motor Manufacturing Turkey firmasında proje mühendisi olarak çalıştı.
- 2022-2023 yılında Stellantis- TOFAS firmasında teknolojik sistemler uzmanı olarak çalıştı.
- 2023-2025 Toyota Motor Manufacturing Turkey firmasında AI-IoT mühendisi olarak çalıştı.
- 2025~ Toyota Motor Corporation firmasında gövde bölümü üretim mühendisi Olarak çalışıyor.

TEZDEN TÜRETİLEN ESERLER:

- Enesalp O.,Muhammed Kürşat U. (2025, 10-3, Mart) - IJISRT25MAR609 “Artificial Intelligence and Image Processing Based Part Feeding Control in a Robot Cell” International Journal of Innovative Science and Research Technology