İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞI DESTEKLİ BİR ENDÜSTRİYEL GÖRÜNTÜ İŞLEME UYGULAMASI

YÜKSEK LİSANS TEZİ

Göktuğ ÜLKÜER

Kontrol ve Otomasyon Mühendisliği Ana Bilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

HAZİRAN 2020

İSTANBUL TEKNİK ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

YAPAY SİNİR AĞI DESTEKLİ BİR ENDÜSTRİYEL GÖRÜNTÜ İŞLEME UYGULAMASI

YÜKSEK LİSANS TEZİ

Göktuğ ÜLKÜER
(504171111)

Kontrol ve Otomasyon Mühendisliği Ana Bilim Dalı

Kontrol ve Otomasyon Mühendisliği Programı

Tez Danışmanı: Prof. Dr. Salman KURTULAN

HAZİRAN 2020

İTÜ, Fen Bilimleri Enstitüsü'nün 504171111 numaralı Yüksek Lisans öğrencisi Göktuğ ÜLKÜER, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "YAPAY SİNİR AĞI DESTEKLİ BİR ENDÜSTRİYEL GÖRÜNTÜ İŞLEME UYGULAMASI" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı : **Prof. Dr. Salman KURTULAN** .....
İstanbul Teknik Üniversitesi

Jüri Üyeleri : **Doç. Dr. Gülay ÖKE GÜNEL** .....
İstanbul Teknik Üniversitesi

Dr. Öğr. Üyesi Ali Evren GÖKSUNGUR .....
Nişantaşı Üniversitesi

Teslim Tarihi : 18 Mayıs 2020
Savunma Tarihi : 24 Haziran 2020

Aileme ve arkadaşlarıma,

ÖNSÖZ

Görüntü işleme uygulamaları hayatımızın birçok alanda kullanılmaktadır ve bu alanlardan biri de endüstriyel otomasyondur. Bu çalışmada endüstriyel ekipmanlar kullanılarak, yapay sinir ağlarının da dahil edildiği bir görüntü işleme uygulaması ele alınmıştır.

Hem lisans hem de yüksek lisans hayatım boyunca hep yanımda olan ve yardımlarını esirgemeyen Prof. Dr. Salman KURTULAN'a teşekkür ederim.

Haziran 2020

Göktuğ ÜLKÜER
(Kontrol ve Otomasyon Mühendisi)

İÇİNDEKİLER

Sayfa

ÖNSÖZ ..... **vii**
İÇİNDEKİLER ..... **ix**
KISALTMALAR ..... **xiii**
SEMBOLLER ..... **xv**
ÇİZELGE LİSTESİ ..... **xvii**
ŞEKİL LİSTESİ ..... **xix**
ÖZET ..... **xxi**
SUMMARY ..... **xxiii**
1. GİRİŞ **1**
2. KULLANILAN DONANIMLAR **3**
2.1 Bilgisayar 3
2.1.1 Bilgisayar mimarisi 4
2.1.2 Endüstriyel bilgisayar 5
2.2 Endüstriyel I/O Aygıtları 6
2.3 Kamera 8
2.3.1 Kameranın mekanikleri 8
2.3.1.1 Diyafram 9
2.3.1.2 Deklanşör 10
2.3.1.3 Işık ölçer 10
2.3.1.4 Lens 10
2.3.1.5 Odak 11
2.3.2 Endüstriyel kameralar 11
3. KULLANILAN YAZILIMLAR VE TEKNOLOJİLER **15**
3.1 TwinCAT 3 15
3.1.1 Gerçek zamanlı çalışma 15
3.1.2 İzole çekirdek ile çalışma 17
3.1.3 Çevrim süresinin aşıldığı durumlar 18
3.2 EtherCAT 18
3.2.1 Çalışma prensibi 19
3.2.2 Protokol 19
3.2.3 Performans 20
3.2.4 Topoloji 20
3.2.5 Senkronizasyon 21
3.2.6 Teşhis 22
3.2.7 İzleme 23
3.2.8 Ağ geçitleri 23
3.3 TwinCAT Vision 24
3.3.1 Gigabit Ethernet (GigE) 25
3.3.2 GigE vision standardı 25
3.3.3 TwinCAT Vision bileşenleri 26

3.3.3.1 Konfigürasyon asistanları ... 26
3.3.3.2 PLC kütüphanesi ... 27
3.3.3.3 GigE Vision sürücüsü ... 27
3.4 Automatic Device Specification (ADS) ... 28
3.5 TwinCAT HMI ... 30
3.5.1 Kullanım senaryoları ... 31
3.5.1.1 Yerel istemci ... 31
3.5.1.2 Çoklu istemci ... 31
3.5.1.3 Çoklu çalışma zamanı ... 33
3.5.1.4 Çoklu sunucu ... 33
3.6 Python ... 34
3.6.1 Tensorflow ... 34
3.6.2 PyAds ... 34
4. YAPAY SİNİR AĞLARI ... 35
4.1 Yapay Nöron Modeli ... 39
4.2 Aktivasyon Fonksiyonu ... 41
4.2.1 Basamak aktivasyon fonksiyonu ... 42
4.2.2 Sigmoid aktivasyon fonksiyonu ... 42
4.2.3 Hiperbolik tanjant aktivasyon fonksiyonu ... 43
4.2.4 ReLu (Rectified linear unit) aktivasyon fonksiyonu ... 43
4.2.5 Softmax aktivasyon fonksiyonu ... 44
4.3 Yapay Sinir Ağı Katmanları ... 45
4.3.1 Giriş katmanı ... 45
4.3.2 Gizli katman ... 46
4.3.3 Çıkış katmanı ... 46
4.4 Yapay Sinir Ağlarında Öğrenme Stratejileri ... 47
4.4.1 Danışmanlı (Supervised) öğrenme ... 47
4.4.2 Danışmansız (Unsupervised) öğrenme ... 47
4.4.3 Destekleyici (Reinforcement) öğrenme ... 48
4.5 Yapay Sinir Ağı Mimarileri ... 49
4.5.1 İleri beslemeli yapay sinir ağı modeli ... 49
4.5.2 Geri beslemeli yapay sinir ağı modeli ... 49
4.6 Hata Düzeltme Tabanlı Öğrenme ... 49
4.7 En Dik Gradyan Azalma Yöntemi ... 51
4.8 Geri Yayılım ... 52
5. ÖRNEK GÖRÜNTÜ İŞLEME UYGULAMASI ... 53
5.1 Uygulamanın Amacı ... 53
5.2 TwinCAT Ortamının Hazırlanması ... 53
5.2.1 Endüstriyel kameranın taratılması ve konfigürasyonu ... 53
5.2.2 Çevrimdışı dosya kaynakları ile çalışma ... 55
5.2.3 PLC projesinin hazırlanması ... 56
5.2.4 Görüntüye uygulanan ön işlem ... 59
5.3 Python İle Yapay Sinir Ağları ... 60
5.3.1 Döngü saysının (epoch) etkileri ... 62
5.3.2 Yığın boyutunun (batch size) etkileri ... 63
5.3.3 Eğitim için kullanılan veri sayısının etkileri ... 64
5.3.4 Gizli katman sayısının etkileri ... 65
5.3.5 Gizli katmandaki nöron sayısının etkileri ... 66
5.4 Ezberlemenin önüne geçmenin yolları ... 67
5.5 Evrişimsel Yapay Sinir Ağı ... 68

5.6 Arayüz Tasarımı 71
6. SONUÇ VE ÖNERİLER 73
KAYNAKLAR 75
EKLER 77
ÖZGEÇMİŞ 89

KISALTMALAR

ADS : Automatic Device Specification
ALU : Arithmetic Logic Unit
AMS : Automatic Machine Specification
ATM : Automated Teller Machine
CAM : Computer Aided Manufacturing
CAT5 : Category 5 cable
CAT6 : Category 6 cable
CNC : Computer Numerical Control
CRC : Cyclic Redundancy Check
DIN : Deutsches Institut für Normung
DMA : Direct Memory Access
ENIAC : Electronic Numerical Integrator and Computer
EtherCAT : Ethernet For Control Automation Technology
Gbit : Gigabit
GigE : Gigabit Ethernet
HMI : Human-Machine Interface
HTML5 : Hyper Text Markup Language
I/O : Input/Output
IEC : International Electrotechnial Commision
IEEE : Institute of Electrical and Electronics Engineers
IP : Internet Protocol
IPC : Industrial Computer
LD : Ladder Diagram
LED : Light Emitting Diode
MAC : Media Access Control
MB : Mega Byte
Mbit : Megabit
Mbps : Megabits per second
NC : Numerical Control
PCI : Peripheral Component Interconnect

PLC **:** Programmable Logic Controller

RAM **:** Random Access Memory

ReLu **:** Rectified Linear Unit

RJ45 **:** Registered Jack - 45

TCP **:** Transmission Control Protocol

TwinCAT **:** The Windows Control and Automation Technology

UDP **:** User Datagram Protocol

USB **:** Universal Serial Bus

WYSIWYG **:** What You See Is What You Get

XAE **:** Extended Automation Engineering

XAR **:** Extended Automation Runtime

YSA **:** Yapay Sinir Ağı

SEMBOLLER

C	**:** Dokunun kapasitansı
f	**:** f-oranı
s	**:** Saniye
V	**:** Volt
W	**:** Watt
θ	**:** Eşik değeri
μ	**:** Mikro
η	**:** Öğrenme oranı
m	**:** Mili

ÇİZELGE LİSTESİ

Sayfa

Çizelge 3.1 : Bilinen ADS portları ve açıklamaları. **29**

ŞEKİL LİSTESİ

Sayfa

Şekil 2.1 : Delikli kart. **3**
Şekil 2.2 : John von Neumann mimarisi. **4**
Şekil 2.3 : Kompakt endüstriyel bilgisayar C6030. **5**
Şekil 2.4 : Kullanılan I/O bileşenleri **7**
Şekil 2.5 : Çeşitli diyafram açıklıkları. **9**
Şekil 2.6 : Çeşitli endüstriyel kameralar. **12**
Şekil 2.7 : Basler acA1300-60gc. **13**
Şekil 2.8 : Ricoh FL-CC0814A-2M. **13**
Şekil 3.1 : Gerçek zamanlı çalışma ayarları. **16**
Şekil 3.2 : PLC tikinin gerçek zamanlı çalışması. **17**
Şekil 3.3 : İzole çekirdek ile çalışma. **17**
Şekil 3.4 : Çevrim süresinde taşma olması durumu. **18**
Şekil 3.5 : Çevrim süresinde üst üste iki kere taşma olması durumu. **18**
Şekil 3.6 : Topoloji çeşitleri. **21**
Şekil 3.7 : EtherCAT sisteminde dağıtılmış saat. **22**
Şekil 3.8 : EtherCAT teşhis yöntemleri. **23**
Şekil 3.9 : TwinCAT Vision'a genel bakış. **24**
Şekil 3.10 : ADS imaj görüntüleme aygıtı. **24**
Şekil 3.11 : GigE Vision. **25**
Şekil 3.12 : TwinCAT 3'ün modüler yapısı ve bazı modülleri. **26**
Şekil 3.13 : TwinCAT Vision PLC kütüphane içeriği. **27**
Şekil 3.14 : ADS konsepti. **28**
Şekil 3.15 : Çoklu haberleşme modeli. **30**
Şekil 3.16 : Farklı ekran boyutlarında TwinCAT HMI. **31**
Şekil 3.17 : Yerel istemci senaryosu. **32**
Şekil 3.18 : Çoklu istemci senaryosu. **32**
Şekil 3.19 : Çoklu çalışma zamanı senaryosu. **33**
Şekil 3.20 : Çoklu çalışma zamanı senaryosu. **33**
Şekil 4.1 : Biyolojik nöron. **35**
Şekil 4.2 : Akson. **36**
Şekil 4.3 : Sinaps. **36**
Şekil 4.4 : Aksondaki aksiyon potansiyeli. **37**
Şekil 4.5 : T = 0 zamanında uyarıldıktan sonra biyofiziksel nöron çıkış sinyalinin gösterimi. **39**
Şekil 4.6 : Yapay nöron. **40**
Şekil 4.7 : Basamak aktivasyon fonksiyonu. **42**
Şekil 4.8 : Sigmoid aktivasyon fonksiyonu. **42**
Şekil 4.9 : Hiperbolik tanjant aktivasyon fonksiyonu. **43**
Şekil 4.10 : ReLu. **43**
Şekil 4.11 : Leaky ReLu. **44**
Şekil 4.12 : Softmax örneği. **44**

Şekil 4.13 : İleri yönlü hesaplamalı ve geri yayılımlı bir ağ mimarisi. **45**
Şekil 4.14 : YSA blok diyagramı. **49**
Şekil 4.15 : İki boyutlu nöron ağırlıkları uzayında hatanın gösterimi. **51**
Şekil 4.16 : Öğrenme oranının etkisi. **52**
Şekil 5.1 : PLC projesinin modülleri. **53**
Şekil 5.2 : Kaynak seçimi. **53**
Şekil 5.3 : Endüstriyel kameranın taratılması. **54**
Şekil 5.4 : Endüstriyel kameranın konfigüre edilmesi. **54**
Şekil 5.5 : Endüstriyel kameranın test edilmesi. **55**
Şekil 5.6 : Endüstriyel kameranın renkli test edilmesi. **55**
Şekil 5.7 : Dosya kaynağı cihazı ekleme. **56**
Şekil 5.8 : Dosyaların eklenmesi. **56**
Şekil 5.9 : Vision kütüphanesinin eklenmesi. **57**
Şekil 5.10 : PLC değişkenleri **57**
Şekil 5.11 : Kamera durumları. **58**
Şekil 5.12 : Durum makinesi. **58**
Şekil 5.13 : Örnek kod parçası. **58**
Şekil 5.14 : Örnek kod parçası. **59**
Şekil 5.15 : Görüntülerin izlenmesi. **59**
Şekil 5.16 : Elyazısıyla yazılmış rakamlar. **60**
Şekil 5.17 : Temsili YSA modeli. **61**
Şekil 5.18 : Döngü sayısına göre başarı oranı ve hata grafikleri. **62**
Şekil 5.19 : Yığın boyutuna göre başarı oranı ve hata grafikleri. **63**
Şekil 5.20 : Veri setinin boyutuna göre başarı oranı ve hata grafikleri. **64**
Şekil 5.21 : Gizli katman sayısına göre başarı oranı ve hata grafikleri. **65**
Şekil 5.22 : Gizli katmandaki nöron sayısına göre başarı oranı ve hata grafikleri. ... **66**
Şekil 5.23 : Eğitim ve test verisinin hata grafiği. **67**
Şekil 5.24 : Ezberleme yapmış YSA'ya ait hata grafiği. **67**
Şekil 5.25 : L1=0.001 ve L2=0.0001 kullanılan YSA'ya ait hata grafiği. **68**
Şekil 5.26 : El yazısı verilerinin değişkenlere atanması. **68**
Şekil 5.27 : Evrişimsel yapay sinir ağı. **69**
Şekil 5.28 : Evrişimsel yapay sinir ağı modeli. **69**
Şekil 5.29 : Yapay sinir ağının derlenmesi ve eğitilmesi. **69**
Şekil 5.30 : Eğitim adımları. **70**
Şekil 5.31 : Modelin başarı grafiği. **70**
Şekil 5.32 : Modelin hata grafiği. **71**
Şekil 5.33 : TwinCAT HMI sayfası. **71**
Şekil 5.34 : TwinCAT HMI sayfası. **72**

YAPAY SİNİR AĞI DESTEKLİ BİR ENDÜSTRİYEL GÖRÜNTÜ İŞLEME UYGULAMASI

ÖZET

Endüstri 4.0 kavramının hayatımıza iyice girmesi ve endüstri tarafından benimsenmesi sonucunda, görüntü işleme uygulamaları endüstride çokça tercih edilmeye başlamıştır; ancak görüntü işleme uygulaması yapmak masraflı olabileceği gibi çok kompleks de olabilir. Özellikle de görüntü işleme algoritmalarını yazmak için yüksek seviye yazılım dili bilmek gerekebilir.

Bu tez çalışmasında yapılacak olan görüntü işleme uygulaması tamamen PLC kullanılarak yapılacaktır ve uygulamanın amacı ise el yazısı ile yazılmış olan rakamları tanıyıp, sınıflandıran ve bu sınıflandırma sonucunu bir arayüz ile kullanıcıya bildiren bir uygulama yapmaktır. Görüntü işleme yazılımı PLC kodu olarak yazılacaktır. Yazılmış olan karakterleri tanıması için de Python yazılım dili ile modellenen ve eğitilen evrişimsel yapay sinir ağı kullanılacaktır.

Tez çalışmasının giriş bölümünde tezde kullanılmış olan yazılım ve donanımlar ile ilgili genel bilgiler verilmiş ve yapılacak olan uygulamanın detayları tartışılmıştır.

Çalışmanın ikinci bölümünde bu tez çalışması sırasında kullanılacak olan fiziksel ekipmanlar yani donanımlar hakkında genel bilgiler verilmiştir. Bu genel bilgilerden sonra da marka ve modelleri belirtilmiştir. Bu donanımlardan bir tanesi ve en önemlisi kullanılmış olan endüstriyel bilgisayardır. Seçilmiş olan cihaz gerçek zamanlı çalışma yapmaya uygundur ve görüntü işleme uygulamaları genellikle işlemciyi yorabilecek veya zorlayabilecek uygulamalar olduğundan, güçlü bir endüstriyel bilgisayar seçilmiştir. Seçilecek olan bilgisayarda bulunması gereken en önemli özelliklerden biri de Gigabit Ethernet bağlantı noktasına sahip olmasıdır çünkü kullanılacak olan endüstriyel kamera GigE Vision standardını desteklemekte ve Gigabit Ethernet üzerinden haberleşmektedir. Uygulamada kullanılmış olan saha giriş çıkış ekipmanlarından bir tanesi de aydınlatma kontrolcüsüdür ve bilgisayar görüşü uygulamalarındaki en önemli faktörlerden biri olan aydınlatmayı hızlı bir şekilde kontrol etmek için EtherCAT uyumlu aydınlatma kontrolcüsü seçilmiştir. Endüstriyel bilgisayar üzerinde EtherCAT portu mevcuttur. Bu sayede de aydınlatma ekipmanlarının kontrolü hızlı ve güvenli bir şekilde yapılabilmektedir. Son olarak bilgisayar Windows 10 işletim sistemine sahiptir ve tüm mühendislik yazılımları içinde çalışmaktadır.

Çalışmanın üçüncü bölümünde ise tez çalışmasında kullanılmış olan teknolojiler ve yazılımlar üzerinde durulmuştur ve bunlar ile ilgili genel bilgiler verilmiştir. İlk olarak, endüstriyel bilgisayarlarda gerçek zamanlı çalışmaya olanak sağlayan TwinCAT yazılımı üzerinde durulmuştur. Bu yazılım Windows işletim sitemi ile birlikte kullanılmaktadır ve elimizdeki Windows tabanlı bilgisayarı, bir PLC gibi kullanmamıza olanak sağlamaktadır. Normal şartlarda işlemcinin bir kısmını işletim sistemine ayırmaktadır; ancak işlemciye ait çekirdeklerin bir veya daha fazlasını izole ederek, tüm işlemci gücünü yapılacak olan endüstriyel uygulamaya ayırmak mümkündür. Bu bölümde izole çekirdek ayarlarının nasıl yapılacağı üzerinde durulmuş ve gerçek zamanlı çalışma konsepti açıklanmıştır. Çevrim süresinin bir kere veya üst üste aşıldığı durumlar grafikler yardımıyla izah edilmiştir. Sonrasında EtherCAT haberleşme protokolü hakkında genel bilgiler verilmiş ve çalışma prensibi

anlatılmıştır. Dağınık saat senkronizasyonu, desteklediği topolojiler, teşhis yetenekleri gibi protokolü öne çıkaran özellikler anlatılmıştır. Bir sonraki adımda ise TwinCAT Vision modülü ele alınmıştır. Bu modül yardımıyla, endüstriyel alanda bilgisayar görüşü kullanmak ve görüntü işleme algoritmalarını uygulamak mümkündür. Modülün içeriği hakkında bilgiler verilmiş ve GigE Vision standardı açıklanmıştır. İlerleyen bölümde de üçüncü parti yazılımlar ile TwinCAT arasında haberleşmeye olanak sağlayan ADS protokolünden bahsedilmiştir ve bu protokolü oluşturan unsurlar üzerinde durulmuştur. Uygulama esnasında operatörün kullanması amacıyla TwinCAT HMI kullanılarak bir arayüz geliştirilmiştir ve bu bölümde de TwinCAT HMI'a hakkında genel bilgiler verilmiş ve kullanım senaryoları üzerinde durulmuştur. Son olarak da yapay sinir ağlarını modellemek ve eğitmek amacıyla kullanılmış olan Python yazılım dili ve bu dil içinde kullanılan Tensorflow ve PyAds kütüphaneleri ile ilgili bilgiler verilmiştir.

Çalışmanın dördüncü bölümünde yapay sinir ağı ile ilgili genel bilgiler verilmiş. Biyolojik nöronlar ele alınarak matematiksel nöronun gelişim sürecinden bahsedilmiştir ve denklemlerle açıklanmıştır. Daha sonraki kısımlarda ise bu yapay sinir ağını oluşturan temel ögelerden bahsedilmiştir. Yapay sinir ağlarının eğitilmesi için kullanılan yöntemler anlatılmış, çeşitli mimariler verilmiştir. Hata azaltma yöntemlerinden biri olan gradyan iniş konseptinden bahsedilmiş ve geri yayılıma değinilmiştir.

Çalışmanın beşinci bölümünde projenin işleyişi ile ilgili bilgiler verilmiştir. Öncelikle TwinCAT ortamında yapılanlar anlatılmış, daha sonrasında ise Python kullanılarak modellenmiş ve eğitilmiş yapay sinir ağları ile ilgili bazı karşılaştırmalar, sonuçlar sunulmuştur. Yapay sinir ağlarında çokça karşılaşılan ezberleme sorunu grafiksel olarak gösterilmiş ve bazı çözüm önerileri sunulmuştur. Bu çözümlerden bir tanesi kullanılarak gösterdiği etki grafiksel olarak gösterilmiştir. Tüm bu işlemler basit bir yapay sinir ağı modelinde yapılmış; ancak esas uygulamada evrişimsel yapay sinir ağı modeli kullanılmıştır. Bu model kendisine verilen veriyi %99 oranında doğru sınıflandırma oranına sahiptir ve objektif fonksiyonu değeri 0,03 olarak hesaplanmıştır, sıfıra yakın olmasın istenilen durumdur. Son olarak da TwinCAT HMI'da tasarlanmış olan ve kameradan veya çevrimdışı dosya kaynağından alınan görüntüleri izleyebildiğimiz, bu görüntülerin PLC içinde uygulanan görüntü işleme algoritmalar sonrasındaki halini de görebildiğimiz ve Python ile haberleşme tetiklerini verip, yapay sinir ağından gelen cevapları görüntüleyebileceğimiz arayüz tanıtılmıştır.

Çalışmanın son bölümünde ise elde edilen sonuçlar paylaşılmıştır ve performansı hakkında yorumlar yapılmıştır. Çalışmada kullanılan donanım ve yazılımlardan bahsedilmiştir.

AN INDUSTRIAL IMAGE PROCESSING APPLICATION WITH ARTIFICIAL NEURAL NETWORK

SUMMARY

With the development of computer technologies, human beings perform almost all their operations on these innovative technologies and enable new methods to be found. That is why the idea that the machine, which was introduced in the 1980s, can think like a human being, has been put forward, and in the 1990s, artificial neural networks technology has accelerated and a great improvement has been observed.

The foundations of the first computational model on which artificial neural networks were laid in an article published by McCulloch and Pitts, who began their research in the early 1940s. In this article, they tried to model an artificial neural network using electrical circuits. Later, in 1954, a model that responded to and adapted to the warnings was created in a network by Farley and Clark. 1960 was the first year of the emergence of the neural computer. The first shortcomings of simple models were noticed in 1963, but the successful results were delayed until the use of theoretical structures in thermodynamics in the development of nonlinear networks in the 1970s and 1980s. 1985 was the year when artificial neural networks were well known and intensive research began. In today's world, artificial neural networks are being utilized in great number of fields, and one of them is image processing applications. Especially, convolutional neural networks are preferred in an image classification application.

As the concept of Industry 4.0 has been introduced into our lives and adopted by the industry, image processing applications have started to be preferred in the industry; However, applying image processing can be very costly or very complex. In particular, it may be necessary to know a high-level software language to write image processing algorithms.

The image processing application to be done in this thesis study will be made entirely using PLC and the purpose of the application is to make an application that recognizes and classifies the numbers written in handwriting and informs the user about this classification result through an interface. Image processing software will be written as PLC code. Convolutional artificial neural network modeled and trained in the Python software language can be used to recognize the characters written.

In this particular thesis, the main goal is to create an application that is capable of recognizing handwritten digits in real-time by using industrial components. In order to achieve real-time computing, powerful industrial computer with Gigabit Ethernet adapter is selected. Also, high quality industrial camera, which can be used with ease thank to GigE Vision standard, is preferred to get the most out of high-speed Gigabit Ethernet interface and easy installation. The quality of the image received from the camera is a high; therefore, image processing algorithms applied on the images become computationally intensive. It is quite significant to select powerful industrial computer to avoid any unnecessary cycle time exceeds in PLCs. Exceeding cycle time

in consecutive sessions is a big problem, and might lead to dangerous situations. Also, it breaks first rule of thumb of real-time computing.

In the introduction part of the thesis study, general information about the software and hardware used in the thesis is given and the details of the application to be made are discussed.

In the second part of the study, general information about the physical equipment that will be used during this thesis is given. After this general information, brands and models are specified. One of this equipment, and the most important one, is the industrial computer used. The selected device is suitable for real-time operation and a powerful industrial computer has been chosen because image processing applications are often applications that can tire or strain the processor. One of the most important features to be found on the computer to be selected is that it has a Gigabit Ethernet port because the industrial camera to be used supports the GigE Vision standard and communicates over Gigabit Ethernet. One of the field IO devices used in the application is the lighting controller and EtherCAT compatible lighting controller has been chosen to quickly control the lighting, which is one of the most significant factors in computer vision applications. EtherCAT port is available on the industrial computer. In this way, the control of the lighting equipment can be done quickly and safely. Finally, the industrial computer has Windows 10 as an operating system and all engineering software were installed inside of it. That's why it is possible to write and download a PLC project by just using the industrial computer. Also, Python algorithms will be running along with the PLC code.

In the third part of the study, the technologies and software used in the thesis study are emphasized and general information about them is given. Firstly, TwinCAT software was focused on. This software allows real-time operation on industrial computers. It is used in conjunction with the Windows operating system and allows us to use the Windows based computer we have, just like a PLC. Under normal conditions, it allocates some of the processor's processing power to the operating system; however, it is possible to isolate all processor power for TwinCAT by isolating one or more of the cores of the processor into the industrial application to be made. This section focuses on how to adjust the isolated core settings and explain the real-time working concept. The situations where the cycle time is exceeded once or in a row are explained with the help of graphics. Afterwards, general information about EtherCAT communication protocol is given and working principle is explained. Protocol highlights such as distributed clock synchronization, topologies it supports, and diagnostic capabilities are described. The next step is the TwinCAT Vision module. With the help of this module, it is possible to use computer vision and apply image processing algorithms in an industrial field. Information about the content of the module is given and GigE Vision standard is explained. The ADS protocol, which enables communication between third party software and TwinCAT, is mentioned in the following section and the components of this protocol are emphasized. An interface has been developed using the TwinCAT HMI for the operator to use during the application, and in this section, general information about the TwinCAT HMI is given and usage scenarios are emphasized. Finally, Python software language used to model and train artificial neural networks and information about Tensorflow and PyAds libraries used in this language are given.

In the fourth part of the study, general information about artificial neural network is given. Biological neurons are discussed and the development process of the

mathematical neuron is explained and explained by equations. In the next sections, basic elements that make up this artificial neural network are mentioned. The importance of the selection of these basic elements were discussed by presenting several plots, which show the impact of these elements. The methods used for training artificial neural networks are explained and various architectures are given. Gradient reduction concept, which is one of the error reduction methods, is mentioned and back propagation is mentioned.

In the fifth part of the study, information about the steps used in the project is given. Firstly, what was done in the TwinCAT environment was explained, and then some comparisons and results about modeled and trained artificial neural networks using Python are presented. The problem of memorization, which is common in artificial neural networks, is shown graphically and some solution suggestions are presented. Using one of these solutions, its effect is shown graphically. All these operations are done in a simple artificial neural network model; however, the convolutional artificial neural network model was used in the project. This model has the probability of classifying the given data 99% correctly and the objective function value has been calculated as 0.03, it is the desired to have loss value as close to zero. Finally, the interface designed in the TwinCAT HMI were introduced, where it is possible to view the images taken from the camera or offline file source. Also, it is possible to give the communication triggers to Python and it displays the answers from coming from the artificial neural network.

In the last part of the study, the results obtained were shared and comments were made about their performance. Hardware and software used in the study are mentioned.

1. GİRİŞ

PLC, otomasyon sistemlerinin kontrolünden sorumlu özel bir kontrolcüdür. PLC'ler temel olarak sensörler ve aktüatörlerle temas halindedir, bu nedenle otomasyon sistemlerini veya proseslerini kontrol edebilir ve izleyebilirler. PLC'lerin sektördeki birçok unsurla yapılandırılabilme özelliği sayesinde otomotiv, kimya, uçak, demiryolu ve benzeri birçok endüstriyel alanda kullanılmaktadır [1]. Bahsedilen bu alanlara ek olarak, görüntü işleme uygulamalarında da artık rahatlıkla PLC kullanmak mümkündür.

Görüntü işleme ve tanıma sistemleri günümüzde gelişen yazılım ve donanım teknolojileriyle birlikte yaygın olarak kullanılmaktadır. Sınıflandırma uygulamaları, robotik, güvenlik sistemleri, askeri çalışmalar, endüstriyel kalite kontrol uygulamaları, tıbbi görüntüleme sistemleri, otonom araç tasarımı, uçuş kontrol sistemleri ve daha pek çok alanda kullanılmaktadır. Özellikle el yazısı ile yazılmış karakterleri tanıma işlemleri çokça karşımıza çıkmaktadır [2].

Bir algoritma veya aritmetik veya mantıksal işlem dizisi kullanmanın neredeyse imkansız olduğu kanıtlanmış çok sayıda görev vardır. Örneğin, birçok denemeye rağmen, el yazısı karakterleri otomatik olarak okuyacak veya herhangi bir konuşmacı tarafından konuşulan kelimeleri bir dilden diğerine çeviren bir makine henüz üretilmemiştir [3]. Ancak günümüzün işlemcilerini ve yapay sinir ağı modellerini kullanaraktan bu işlemleri yapmak mümkündür.

Bu tez çalışmasında Windows tabanlı bir endüstriyel bilgisayar gerçek zamanlı çalışma prensibini esas alarak, endüstriyel bir kameradan görüntü alacak, PLC içinde görüntü algoritmalarını koşturduktan sonra da Python kodunda modellenmiş ve eğitilmiş evrişimsel yapay sinir ağır yardımıyla sınıflandırılacaktır. Bu sınıflandırmanın sonucu da kullanıcı arayüzü ile sunulacaktır.

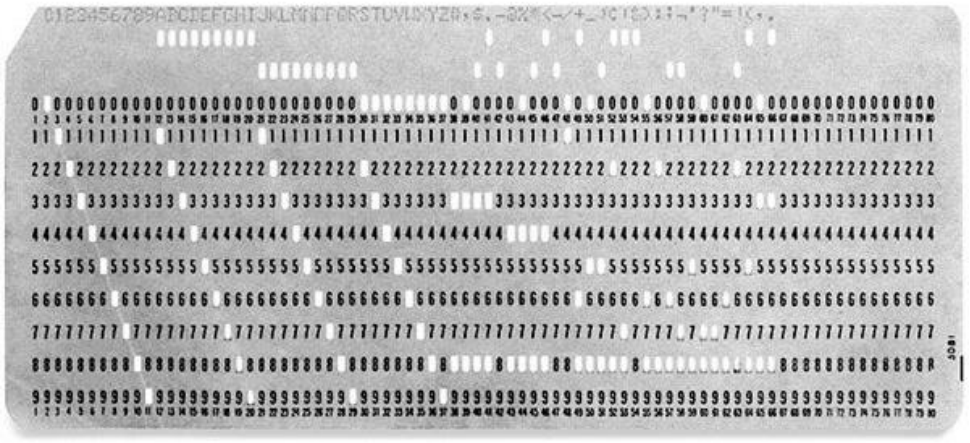
2. KULLANILAN DONANIMLAR

2.1 Bilgisayar

Bilgisayar, aritmetik veya mantıksal operasyonları otomatik olarak yapabilen, gerektiğine veri depolayan, gerektiğinde de bu verileri kullanıcıya sunan cihaza denir [4]. Bilgisayarlar, program isimli genel işlem kümeleri sayesinde, hayatımızın birçok alanındaki çeşitli görevleri yapabilmektedirler.

Geçmişe bakıldığında, bilgisayarlar farklı şekillerde karşımıza çıkmaktadır. 1950'lerde geniş oda büyüklüğünde ve çok ciddi miktarda enerji tüketen bilgisayarlar, yarı iletkenler ve küçük alanlara sığdırılabilen devreler sayesinde bir kol saatine sığabilen ve ufak bir pil ile çalışabilen duruma gelmişlerdir.

Öncelerde 'bilgisayar' olarak bilinen aygıtlar, günümüz ölçütlerinde bu tanımlama ile kullanılmayı hak etmemektedir. İlk dönem bilgisayarları arasında sayı boncuğu (abaküs) ve Antikitera Makinesi gibi nesneler vardı ve bu nesnelerden dolayı, bilgisayar adı, hesaplama sürecini kolaylaştıran nesneler için kullanılan bir isim olarak karşımıza çıkmaktadır. Bu nesnelerin bilgisayar tanımına uymama nedeni ise yazılımlanabilir olmamalarıdır; ancak Joseph Marie Jacquard'ın 1801 yılında ve dokuma tezgahlarındaki işlemleri otomatikleştirme etmek için ürettiği delikli kartlar (Şekil 2.1), kısıtlı olmalarına rağmen, yazılımlanabilmenin ilk izlerinden sayılır.



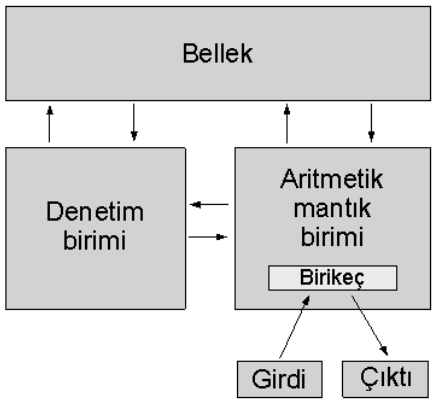
Şekil 2.1 : Delikli kart.

Charles Babbage tarafından 1837 tasarlanan ancak maddi nedenlerden dolayı hayata geçirilemeyen ve "Analitik makine" olarak isimlendirilen makine, ilk tam olarak yazılımlanabilen makinesel bilgisayardır.

1960'lı yıllarda 1950'ler boyunca kullanılan ve boşluk tüplerine dayalı bilgisayarlar yerine daha hızlı ve ekonomik olan transistör tabanlı bilgisayarların kullanımı yaygınlaşmıştır ve bu bilgisayarların seri üretimine geçilmiştir. 1990'lı yıllarda da Internet'in gelişimi ile bilgisayarlar neredeyse her hanede olan birer aygıt haline gelmiştir.

2.1.1 Bilgisayar mimarisi

Bilgisayar mimarisi terimi, bilgisayar bileşenlerinin iç yapıları ve birbirleri arasındaki haberleşmesi ile ilgilidir. Günümüz bilgisayarlarının mimarisinin modelinin geliştiricisi, ünlü matematikçi, bilgisayar bilimcisi ve ENIAC'ın gelişim sürecinde de çalışmış olan John von Neumann'dır. Bilgisayarın iç yapası ve çalışma prensibini kapsayan bu mimariye de "John von Neumann mimarisi" (Şekil 2.2) denmektedir ve bu model mantıksal olarak bilgisayar sistemini kapsamlı bir şekilde tanımlamıştır.



Şekil 2.2 : John von Neumann mimarisi.

John von Neumann mimarisi 4 ana parçadan oluşmaktadır ve bu parçalar aşağıdaki gibidir.

- Aritmetik Mantık Birimi (ALU)
- Denetim birimi
- Bellek
- Giriş ve çıkış birimleri (I/O)

Matematiksel ve mantıksal işlemleri yapan birim ALU'dur ve neredeyse tüm bilgisayarların yapıtaşıdır. Bilgilerin işlenmesini sağlayan birim ise denetim birimidir. Hafıza, bilgi saklamaya yarayan birimdir. Giriş ve çıkış birimleri ise bilgisayar ile insanlar arasındaki iletişimi sağlayan birimdir.

John von Neumann mimarisi dışında kuantum mimarisi, vektör mimarisi, Numa mimarisi, Harvard mimarisi gibi mimariler de vardır; ancak temel olarak iki bilgisayar mimarisi üzerine yoğunlaşılmıştır. Bunlardan bir tanesi John von Neumann mimarisi ve diğeri de Harvard mimarisidir.

2.1.2 Endüstriyel bilgisayar

Bir masaüstü bilgisayar ile endüstriyel bilgisayar mikroişlemci, RAM tipi, performans, arabirim bağlantı noktaları ve depolama ortamı gibi teknik özellerinin çoğu aynıdır; ancak endüstriyel bilgisayarları, normal günlük bilgisayarlardan ayıran en büyük özelliği, zorlu çevresel ortamlarda kullanılmaya uygun olmalıdır. Çok sıcak ortamlarda çalışabilme, sıvı veya toz korunması, titreşim korunması, elektromanyetik uyumluluk vb. gibi özellikleri sağlamak zorundadır ve bu yüzden de günlük hayatta kullanılan bilgisayarlara göre daha maliyetlidirler.

Projede kullanılan IPC, Beckhoff markasına ait C6030 (Şekil 2.2) modelli IPC'dir.



Şekil 2.3 : Kompakt endüstriyel bilgisayar C6030.

Yapılacak uygulama da işlemci gereksinimi yüksek olduğu ve Gigabit Ethernet portu kullanılması gerektiği için bu endüstriyel bilgisayar seçilmiştir. Bunun dışında, arka planda yapay sinir ağlarını gerçeklemek için Python kodu koşturulacağından, seçilecek olan endüstriyel bilgisayarın işletim sisteminin Windows olması, kolaylık sağlayacaktır.

2.2 Endüstriyel I/O Aygıtları

I/O'nun tanımını yapmak gerekirse, en temel tanımında, giriş ve çıkışları temsil eder. Endüstriyel otomasyon sektöründe, girişler ve çıkışlar, butonlar, anahtarlar ve sensörler ile ilgilidir. Örneğin, bir kişi bir basma düğmesine basarak bir giriş sinyali oluşturabilir. Sonuç olarak, bir ışık yanabilir ve bu da çıkış sinyalidir. Tahmin edebileceği üzere, endüstriyel ekipmanlar, değişen karmaşıklık derecelerine göre yüzlerce giriş ve çıkış sinyaline sahip olabilir. Bir I/O sistemi, bir makinedeki cihazlardan gelen sinyalleri gruplandırmanıza ve endüstriyel ekipmanınızın tamamını izlemenize ve kontrol etmenize yardımcı olmaktadır.

Sensörler, solenoidler veya sürücüler gibi giriş/çıkış (I/O) aygıtları, bir kontrol kasasının içine veya dışına monte edilebilir. Bu giriş/çıkış aygıtlarının pano içindeki yerleri ve kablolaması her zaman bir tasarım konusudur.

I/O sistemleri yıllar içinde otomasyon teknolojisi ile birlikte gelişmiştir. Yine de çoğu zaman saha cihazları hala bir PLC kabinindeki giriş ve çıkış modüllerine kablo bağlantılı bir kontrol kasasındaki terminal şeritlerine bağlanır. Bu raf tabanlı ve benzer I/O modülleri iyi bir başlangıç noktasıdır ve yıllardır kullanılmaktadır. Bununla birlikte, bugün I/O sistemleri birçok biçimde olabilir.

Bazıları kablo demetlerini azaltır ve sonlandırma noktasını muhafazanın dışına taşır ve diğer formlar dijital iletişim kablosuyla bağlanır. Tüm yöntemler hala geçerlidir ve birçok durumda tüm yöntemler tek bir otomasyon sisteminde kullanılır.

Zamanında, endüstriyel makinelerin otomasyonunu sağlamak için kullanılan, rölelerle dolu kontrol kabinleri vardı. Zamanla rölelerden oluşan bu kontrol kabinleri, PLC'lerle değiştirildi, ancak sahadaki I/O'lara giden tüm kablolar yerinde kaldı.

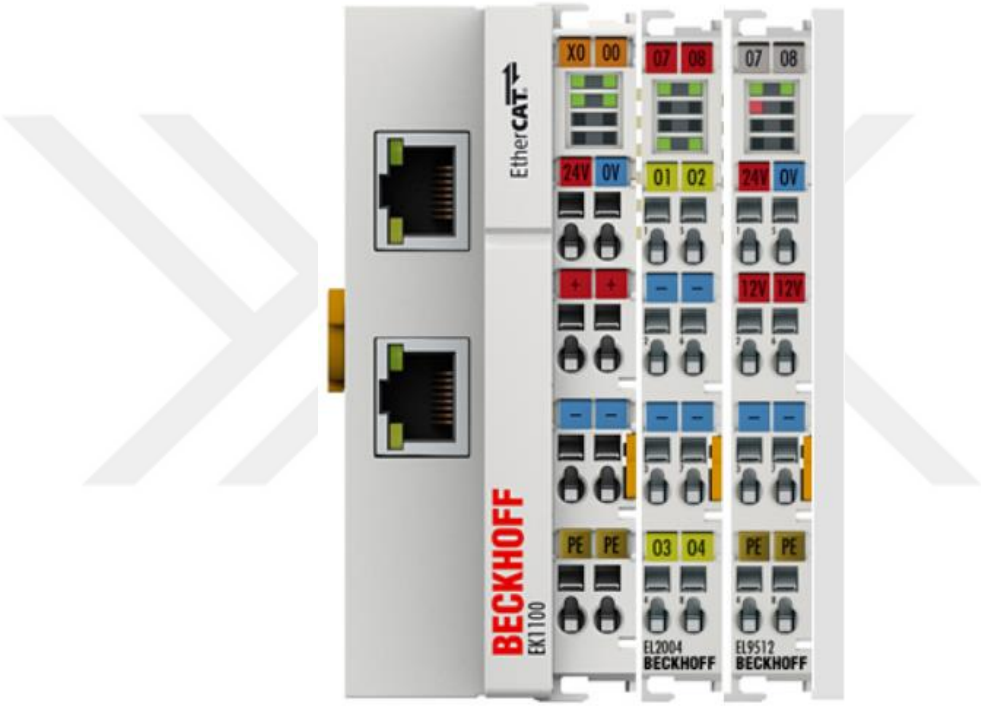
Röleleri PLC'lerle değiştirmek çok önemliydi ve raf tabanlı I/O modülleri hiç bu kadar popüler olmamıştı. Röle kontrol kabinlerinden kurtulmak ve bir PLC kurmak nispeten kolaydı. PLC rafları, rölelerden daha az yer kaplıyordu ve rafa monte edilmiş I/O modülleri sahada bulunan cihazlara kolaylıkla bağlanabiliyordu.

O zaman ve bugün olduğu gibi, binlerce giriş/çıkış noktalarını destekleyen üreticiye özgü haberleşme modülleri ve protokolleri ile genişletme rafları eklenebilir. Yıllar geçtikçe, bu raf tabanlı I/O'lar, modül başına I/O noktalarının sayısı iki kat artarken boyutları da küçülmüştür. Tüm saha cihazlarınızı ayrı bir kablo kullanarak bir kontrol

kabini bölmesi üzerinden terminal bloklarına aktarmak geçerli bir yöntemdir, entegrasyon sırasında kablolamaya giden iş gücü çok olmasına rağmen en kolay kablolama yöntemlerine sahiptir.

Kontrol kabini boyutunu küçültmek ve panoya giren ve çıkan kablo demetlerinin sayısını azaltmak için, sensör sonlandırma noktaları onlarca yıldır ana kontrol kasalarının dışına taşınmaktadır. Bağlantı kutuları, uzak I/O ve saha veri yolu (fieldbus) I/O üç örnektir.

Bu projede Beckhoff markasına ait I/O bileşenleri (Şekil 2.4) kullanılacaktır.



Şekil 2.4 : Kullanılan I/O bileşenleri

Kullanılan ürünler sırasıyla, EK1100, EL2004, EL9512 ve EL9011'dir. EK1100 bir kuplördür ve EK1100 kuplörü EtherCAT'i EtherCAT terminallerine bağlar. Bir istasyon EK1100 kuplörü, herhangi bir sayıda EtherCAT terminali ve bir sonlandırıcı terminalden oluşur. Bu sonlandırma terminali de EL9011'dir. Ethernet 100BASE-TX'den geçen telegramları, "E-bus" olarak adlandırılan sinyal tipine dönüştürür.

Kuplör üzerindeki RJ45 portlarından üst taraftaki ağa bağlanmak için kullanılır ve bu projede C6030'a bağlanacaktır. Alt RJ45 soketi ise aynı başka EtherCAT cihazlarını bağlamak için kullanılabilir ve bu sayede topoloji genişleyebilir. EtherCAT ağında EK1100 kuplörü, Ethernet sinyal aktarım bölümünde (100BASE-TX) herhangi bir yere monte edilebilir; ancak ağ anahtarlarına doğrudan bağlanmamalıdır.

EL2004 ise bir dijital çıkış terminalidir ve otomasyon ünitesinden gelen ikili kontrol sinyallerini elektrik izolasyonlu bir şekilde proses seviyesindeki aktüatörlere bağlar. Bu EtherCAT terminali, sinyal durumunu bir LED ile gösterir.

EL9512 güç kaynağı ünitesi terminali, 24 V DC giriş voltajından 12 V DC çıkış voltajı üretir. Çıkış voltajına terminalden erişilebilir. Yanındaki EtherCAT terminalleri de güç kontakları üzerinden bu voltajla beslenir. Güç LED'leri terminallerin çalışma durumunu gösterir; aşırı akım LED'i kısa devreyi veya aşırı akımı gösterir. Giriş gerilimi ve çıkış gerilimi elektriksel olarak izole değildir.

2.3 Kamera

Kamera, görüntüleri kaydetmek için kullanılan optik bir araçtır. En temelde, kameralar ışığa duyarlı bir yüzeyde (genellikle fotoğraf filmi veya dijital sensör) bir görüntü yakalamak için ışığın içeri girmesine izin veren küçük bir deliğe (diyafram) sahip kapalı kutulardır. Kameralar, ışığın ışığa duyarlı yüzeye nasıl düştüğünü kontrol etmek için çeşitli mekanizmalara sahiptir. Lensler kameraya giren ışığı odaklar, diyaframın boyutu daha fazla veya daha az ışığın kameraya girmesini sağlamak için genişletilebilir veya daraltılabilir ve bir deklanşör mekanizması fotoğrafa duyarlı yüzeyin ışığa maruz kalma süresini belirler.

Fotoğraf makinesi, fotoğraf sanatının ana aracıdır ve çekilen görüntüler daha sonra dijital görüntüleme veya fotoğraf baskısı sürecinin bir parçası olarak çoğaltılabilir.

Kamera kelimesi, karanlık oda anlamına gelen “camera obscura” isminden gelmektedir ve harici bir gerçekliğin imajını düz bir yüzeye yansıdan orijinal cihazın Latince ismidir. Kameranın çalışması ise insan gözünün işlevine çok benzer. İlk kalıcı fotoğraf 1825 yılında Joseph Nicéphore Niépce tarafından çekildi.

2.3.1 Kameranın mekanikleri

Bir kamera, genellikle insan gözüyle görülebilen spektrumdaki ışık fotonlarını yakalar; ancak genel olarak elektromanyetik spektrumun diğer kısımlarından da olabilir [5].

Tüm kameralar temelde aynı tasarımı kullanır: ışık, yakınsak veya dışbükey bir mercekle kapalı bir kutuya girer ve görüntü ışığa duyarlı bir ortama kaydedilir. Bir deklanşör mekanizması da ışığın kameraya girebileceği süreyi kontrol eder. [6]

Çoğu kameranın kaydedilecek sahneyi gösteren bir vizörü vardır. Ek olarak da görüntüyü odaklama ve kamera içine alınan ışığı kontrol etme yeteneği vardır. Kamera içine alınan ışığın kontrolü sayesinde de görüntünün parlaklığı istenilen oranda ayarlanabilir. [7]

2.3.1.1 Diyafram

Bazen diyafram veya iris olarak adlandırılan açıklık, ışığın kameraya girdiği açıklıktır. Tipik olarak lensin içinde bulunan bu açıklık filme çarpan ışık miktarını kontrol etmek için genişletilebilir veya daraltılabilir. Diyaframın merkezindeki deliği küçültmek veya genişletmek için birlikte dönen ve üst üste binen plakaların veya bıçakların hareketleri kullanılır. Açıklığın çapı manuel bir şekilde, tipik olarak kamera gövdesi veya mercek üzerindeki bir kadrandan ayarlanarak veya dahili bir ışık ölçerin etkilediği hesaplamalara göre otomatik olarak ayarlanabilir.

Açıklığın boyutu, genellikle "f-stop" olarak adlandırılan standart artışlarla ayarlanır. $f/1{,}4$ ila $f/32$ arasında standart artışlarla 1,4, 2, 2,8, 4, 5,6, 8, 11, 16, 22 ve 32 gibi değerler almaktadır. Sayılar arttıkça, her bir artış kameraya giren ışık miktarını yarıya indirir. Tersine, sayı ne kadar düşükse, açıklık o kadar büyük olur ve bu nedenle kameraya daha fazla ışık girmesine neden olur (Şekil 2.5).



Şekil 2.5 : Çeşitli diyafram açıklıkları.

Düşük *ƒ* duraklarındaki geniş açıklık, odak aralığını daraltır, böylece ön plana odaklanırken görüntünün arka planı bulanık olur veya bunun tersi de geçerlidir. Bu "alan derinliği" diyafram kapandıkça artar, böylece kameradan farklı mesafelerde olan nesnelerin her ikisi de odakta olabilir; diyafram en dar olduğunda hem ön hem de arka plan keskin odaktadır.

2.3.1.2 Deklanşör

Deklanşör, diyaframla birlikte kameraya giren ışık miktarını kontrol etmenin iki yolundan biridir. Deklanşör, ışığa duyarlı yüzeyin ışığa maruz kalma süresini belirler. Deklanşör açılır, ışık kameraya girer ve filmi veya sensörü ışığa maruz bırakır, ardından deklanşör kapanır.

2.3.1.3 Işık ölçer

Çoğu modern kamerada, kameraya giren ışık miktarı kamera içindeki bir ışık ölçer kullanılarak ölçülür. Mercekten alındığında, bu okumalar ışığa duyarlı bir yarı iletken paneli kullanılarak alınır. En iyi ışığa maruz kalma ayarlarının hesaplanmasına yardımcı olmak için kullanılırlar. Bu ayarlar genellikle, fotoğraf makinesinin mikroişlemcisi tarafından otomatik olarak belirlenir. Işık ölçerden okunan değerlerle birlikte, diyafram açıklığı ayarı, ışığa maruz kalma zamanı ve sensör hassasiyeti, kamera içine alınan optimal ışığın hesaplanmasında kullanılır.

2.3.1.4 Lens

Bir kameranın merceği, nesneden gelen ışığı yakalar ve onu sensörün odak noktasına getirir. Lensin tasarımı ve üretimi, çekilen fotoğrafın kalitesi için kritik öneme sahiptir. 19. yüzyılda kamera tasarımındaki teknolojik devrim, okuma camlarından mikroskoplara kadar çok çeşitli optik cihazlarda kullanılan modern lens üretimi için büyük faydalar sağlayan optik cam üretiminde ve lens tasarımında devrim yarattı.

Kamera lensleri çok çeşitli odak uzunluklarında üretilir. Aşırı geniş açı, standart, orta telefoto gibi menzillere sahiptir. Her lens belirli bir fotoğraf türüne uygundur. Aşırı geniş açı, bir binanın geniş bir görüntüsünü yakalama kapasitesine sahip olduğu için mimari için tercih edilebilir. Normal lens, genellikle geniş bir diyaframa sahip olduğundan, sokak ve belgesel fotoğrafçılık için sıklıkla kullanılır. Telefoto lens spor ve vahşi yaşam için kullanışlıdır ancak kamera titremesine daha duyarlıdır. [8]

2.3.1.5 Odak

Fotoğrafik lenslerin optik özellikleri nedeniyle, yalnızca kameradan sınırlı mesafelerdeki nesneler net bir şekilde gözükecektir. Bu aralığı ayarlama işlemi, kameranın odağını değiştirme olarak bilinir. Bir kamerayı doğru şekilde odaklamanın çeşitli yolları vardır. En basit kameralar sabit odağa sahiptir ve objektiften genellikle 3 metre veya daha uzak bir mesafedeki her şeyin makul odakta olmasını sağlamak için küçük bir diyafram ve geniş açılı lens kullanır. Sabit odaklı kameralar genellikle tek kullanımlık kameralar gibi ucuz türlerdir.

Telemetre kameraları, kameranın üst kısmındaki bir çiftli paralaks ünitesi aracılığıyla nesnelere olan mesafenin ölçülmesine izin vererek odağın hassas bir şekilde ayarlanmasını sağlar. Tek lensli refleks kameralar, fotoğrafçıya görüntüyü bir zemin camına veya plastik mikro prizma ekranına yansıtmak için objektif lens ve hareketli bir ayna kullanarak odağı ve kompozisyonu görsel olarak belirlemesine olanak tanır. Çift lensli refleks kameralar, kompozisyon ve odaklama için paralel bir gövdede objektif lens ve odaklama lens birimi kullanır. Modern kameralar genellikle kamerayı çeşitli yöntemlerle otomatik olarak odaklamak için otomatik odaklama sistemleri sunar. [9]

2.3.2 Endüstriyel kameralar

Endüstriyel kameralar günümüzde çeşitli alanlarda, özellikle üretim izleme ve bir dizi karmaşık ölçüm görevlerinde kullanılmaktadır. Kalite kontrol, endüstriyel görüntü işlemeye güçlü bir şekilde dayanan bir başka alandır. Dijital endüstriyel kameralar genellikle tatil fotoğrafları için kullanılan standart dijital kameralardan daha sağlamdır. Ortamın sıcaklığının yüksek olması gibi birçok dış etkene rağmen, endüstriyel kameralar sorunsuz bir şekilde çalışmaktadır.

Endüstriyel kameralardaki görüntü kalitesi genellikle "normal" bir kamerada bulunandan daha yüksektir, çünkü özel ölçüm görevleri için sıklıkla ayrıntılı görüntüler gereklidir. Vizör, deklanşör ve flaş gibi dijital kameranın tanıdık öğeleri endüstriyel kameralarda bulunmaz. Bu kameralar bir arabirim kullanılarak bilgisayara bağlanır. Kamera bir I/O (giriş / çıkış) kablosu kullanılarak tetiklenir. Bağlı PC'deki yazılım görüntüyü yakalamak için sinyal gönderir. Mecazi anlamda, buna "deklanşör" denebilir.

Endüstriyel kameralar birçok alanda kullanılabilir. Fabrika otomasyonundaki kullanımını ele alacak olursak; yarı iletken denetimi, gıda denetimi veya posta sınıflandırması için endüstriyel kameraların kullanıldığını görülebilir. Otomatik üretim süreçlerinin izlenmesine yardımcı olabileceğinden, özellikle fabrika otomasyonunda sıklıkla kullanılırlar. Örneğin baskı denetimini ele alacak olursak. Burada, tüm basılı görüntülerin belirtilen kalite standardını karşılayıp karşılamadığını kontrol etmek için bir kamera ve yazılım kullanılabilir. Tıp alanına bakılacak olursa, teknik gelişmeler, tıp alanında modern dijital kameraların kullanımına da izin vermiştir. Endüstriyel kameralar, otantik, yüksek çözünürlüklü canlı görüntüler ürettikleri için mikroskopide de kullanılır. Bunun ötesinde, endüstriyel kameralar diş tarayıcılarında, çeşitli laboratuvar uygulamalarında ve hatta fizik tedavi için de kullanılabilir. Trafik ve ulaşım alanlarında da endüstriyel kameralar ile karşılaşmak mümkündür. Kameraların görme gücü modern ve akıllı trafik sistemlerinde kullanılmaktadır. Plaka tanımadan, trafik kuralları ihlallerinin algılanmasına kadar uzanan geniş bir kullanım alanına sahiptir. Ayrıca perakende sektöründe de karşımıza çıkmaktadır. Perakende pazarı için tipik uygulama örnekleri arasında kasa sistemleri (otomatik ödeme) kameraları, ATM'ler ve otomatlar sayılabilir. ATM'lerle birleştirildiğinde, kameralar hem müşteri hem de bankanın kendisi için daha fazla güvenlik sunar. Tam otomatik kendi kendine ödeme sistemleri, müşterilerin ödeme sırasında bekleme sürelerini azaltmak için ve müşterilerin kendi satın alma işlemlerini başlatma olanağını sağlar. Kameralar malları tanımlamak için kullanılır. Bunun ötesinde, bu süreçleri izlemek ve gerekli güvenliği sağlamak için kameralar da kullanılabilir (Şekil 2.6).



Şekil 2.6 : Çeşitli endüstriyel kameralar.

Bu projede kullanılacak olan endüstriyel kamera Basler markasının acA1300-60gc modelli ürünüdür (Şekil 2.7).



Şekil 2.7 : Basler acA1300-60gc.

Kullanılan lens ise Ricoh markasına ait FL-CC0814A-2M modelli lensidir (Şekil 2.8).



Şekil 2.8 : Ricoh FL-CC0814A-2M.

Lens seçimi yapılırken yapılacak uygulamaya göre seçim yapılmalıdır.

3. KULLANILAN YAZILIMLAR VE TEKNOLOJİLER

3.1 TwinCAT 3

Otomasyon mühendisliğinde sürekli artan bir öneme sahip olan entegre edilmiş geliştirme ortamları, hangi sistemi kullanacağı ile ilgili karar alacak bir kullanıcı için çok önemli bir kriterdir. Ayrıca arttırılmış mühendislik verimliliği de göz önünde bulundurulmalıdır. Beckhoff'un TwinCAT 3 yazılımı, Windows tabanlı herhangi bir endüstriyel bilgisayarı, gerçek zamanlı çalışabilen kontrolcüye dönüştürür [10] ve IEC 61131-3 PLC yazılımı, NC yazılımı ve CNC yazılımı gömebilmeyi mümkün kılar. TwinCAT 3, dünya çapında birçok uygulamada denenmiş ve test edilmiş, tanıdık TwinCAT 2 otomasyon yazılımının tutarlı teknolojik gelişiminin sonucudur. Nesneye yönelik IEC 61131-3 uzantılarına ek olarak, bilgi teknolojileri dünyasının C ve C ++ dilleri artık TwinCAT kullanıcıları tarafından da kullanılabilir. Matlab/Simulink'in entegrasyonu da bilimsel uygulamaların yolunu açar. Farklı dillerde yazılmış modüller ortak bir eXtended Automation Runtime'da (XAR) çalışır. Bu, çok çekirdekli teknoloji kullanımı ve 64 bit işletim sistemlerinin desteği ile zorlu gerçek zamanlı koşullara bir cevap sağlar. TwinCAT 3 modüllerinin uygulanması ve tüm sistemin yapılandırılması, Microsoft Visual Studio tabanlı eXtended Automation Engineering ortamında (XAE) gerçekleşir.

TwinCAT 3, özellikle iki kullanıcı grubu için idealdir: Birincisi, IEC 61131-3 uyumlu yazılım ortamlarına aşina PLC programcıları; diğer grup ise C++ kullanarak endüstriyel uygulama yazmak isteyen kullanıcı gruplarıdır. LD dili gibi alışılageldik programlama dili ile sezgisel yöntemler kullanılabileceği gibi, kalıtım mekanizmasıyla nesneye dayalı programlama yapmak da mümkündür.

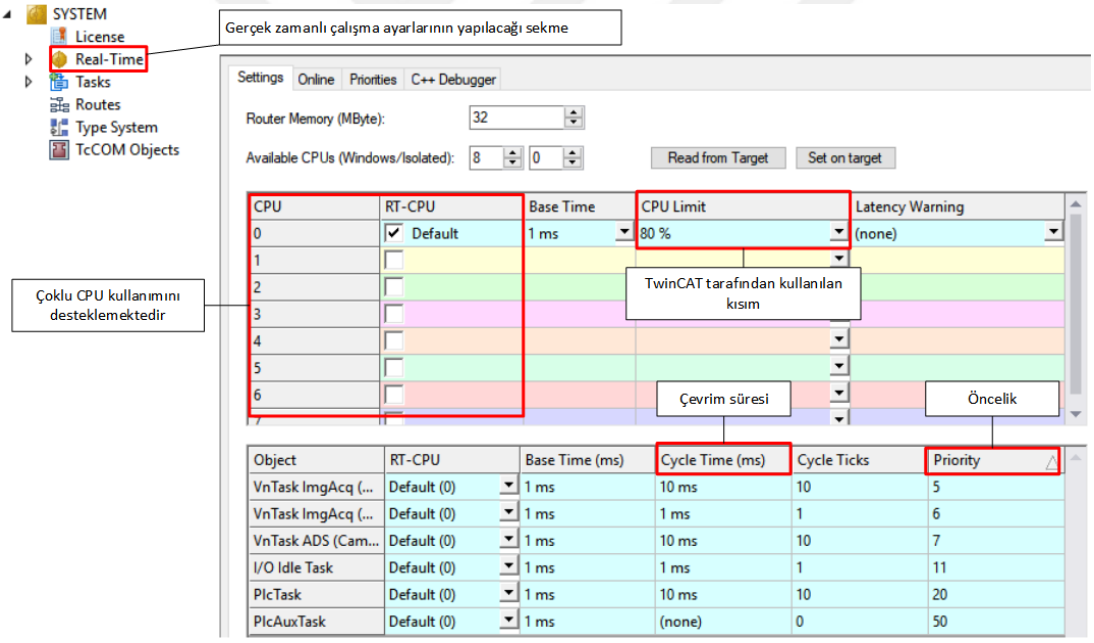
3.1.1 Gerçek zamanlı çalışma

DIN 44300 standardına göre, gerçek zamanlı çalışma şu şekilde tanımlanır: Gerçek zamanlı çalışma, veri işlemenin sürekli olduğu ve bu işlemlerin sonucunun her zaman belirlenen süre içinde elde edildiği çalışma şeklidir.

Başka bir deyişle, bir uygulama programının çıktı değerleri, tanımlanan ve garanti edilen bir süre içinde kullanılabilir. Tanımlanan bu zamana çevrim süresi (Cycle time) denir.

Ayrıca, TwinCAT 3 Real-Time, yüksek performanslı ve esnek/genişletilebilir kontrol platformlarına yönelik artan talepleri karşılamak için çok çekirdekli destek de sunmaktadır. Kullanılabilir çekirdekler yalnızca TwinCAT için kullanılabilir veya Windows ile paylaşılabilir (Şekil 3.1).

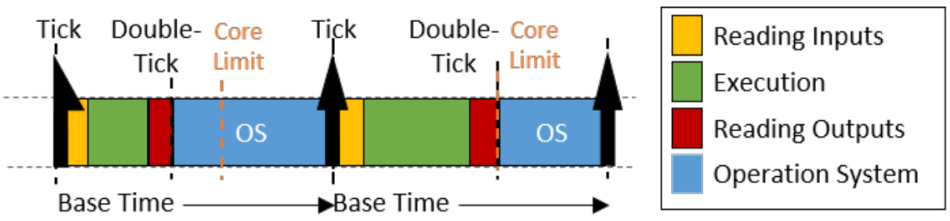
TwinCAT 3 gerçek zamanlı işletim çift tik yöntemiyle çalışır. Tik (Tick) ise bilgisayardaki işlemci çekirdeğinin en küçük saat darbesine verilen addır. Çift tik yöntemi hem gerçek zamanlı moda geçmenin hem de geri dönmenin bir kesinti tarafından tetiklendiği anlamına gelir. Gerçek zamanlı moda geçerken yapılan kesme aynı zamanda programlamayı da başlatır. Ayarlanabilir bir süre sonra, ancak ayarlanan çevrim süresinin %90'ından fazla olmamak kaydıyla, TwinCAT yalnızca paylaşılan çekirdeklerde gerçek zamanlı olmayan moda geri döner, böylece işletim sistemi, donanım işlevleri vb. için gerekli işlemleri yapar. Yalıtılmış çekirdekler bir istisnadır, yani yalıtılmış çekirdeklerde işletim sistemine ait herhangi bir işlem yapılmaz.



Şekil 3.1 : Gerçek zamanlı çalışma ayarları.

Zamanlama (Scheduling), tanımlanan çevrim süresine ve tanımlanan önceliğe bağlı olarak, tek tek görevlerin işlem sırasını ve işlem süresini belirleyen işlemi ifade eder. İşlem süresine sıkı sıkıya bağlı kalmak, yukarıda açıklanan gerçek zamanlı uyumluluğun garanti edilmesini sağlar. Tüm gerçek zamanlı çekirdeklerdeki zamanlamalar, birbirlerinden bağımsız olarak ayarlanır. Bu, farklı çekirdeklerde çalışan gerçek zamanlı görevlerin birbirini etkilememesini garanti eder.

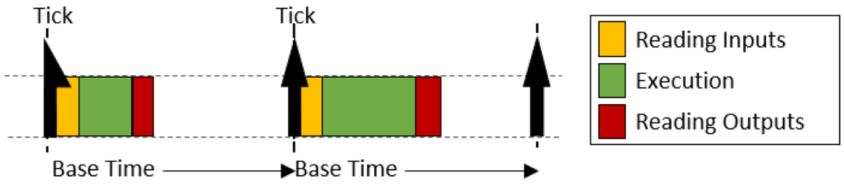
Şekilde bir PLC görevinin tik içinde nasıl çağırıldığı gösterilmektedir. Gerçek zamanlı işaretleme tikinden sonra PLC görevi zamanlayıcı tarafından çağrılır (Şekil 3.2). Bu, mevcut giriş değerlerini PLC uygulaması (giriş güncellemesi) için kullanılabilir hale getirir, ardından uygulama programının işlenmesi (çevrim güncellemesi). Son olarak sonuçlar çıkışlara yazılır (çıkış güncellemesi). Bu tamamlandıktan sonra, cihaz gerçek zamanlı olmayan moda geçer. Şekilde gösterildiği gibi, kullanıcı programının yürütme süresi, hangi kodun yürütüldüğüne bağlı olarak değişebilir. Böylece çıkışların yazıldığı zaman da değişir. Bir haberleşme sisteminin hangi görevi kullandığına bağlı olarak, bu, veri yolu telgramlarının gönderilmesinde kıpırdamaya (jitter) neden olabilir.



Şekil 3.2 : PLC tikinin gerçek zamanlı çalışması.

3.1.2 İzole çekirdek ile çalışma

TwinCAT, belirli bir zamanda gerçek zamanlı olmayan moda geri dönmek için çift tikli bir prosedür kullanır. Gerçek zamanlı mod ile gerçek zamanlı olmayan mod arasında geçiş yaparken, önceki işlem durumu geri yüklenir. Geri yükleme, gerçek zamanlı ve gerçek zamanlı olmayan programların belleği ve özellikle önbelleği ne kadar yoğun kullandığına bağlı olarak biraz zaman alabilir (Şekil 3.3).

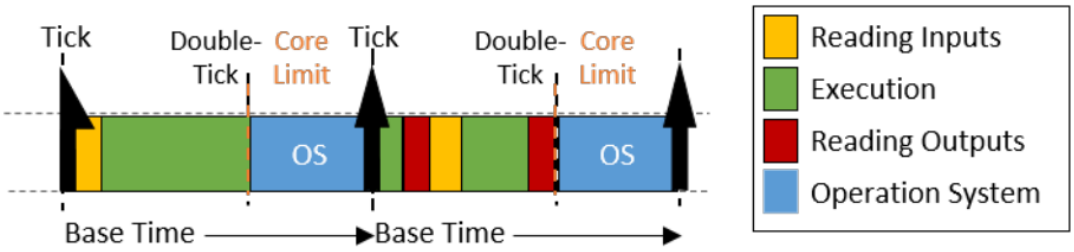


Şekil 3.3 : İzole çekirdek ile çalışma.

Bu geçici etkileri ortadan kaldırmak için TwinCAT 3, çekirdeklerin işletim sisteminden izole edilmesini sağlar. Bu, geri dönme ihtiyacını ortadan kaldırır ve "eski" işlem durumunun geri yüklenmesi ile ilişkili zaman etkilerinden kaçınarak gerçek zamanlı kullanıcı programı için daha fazla işlem süresi sağlar, kıpırdamayı azaltır.

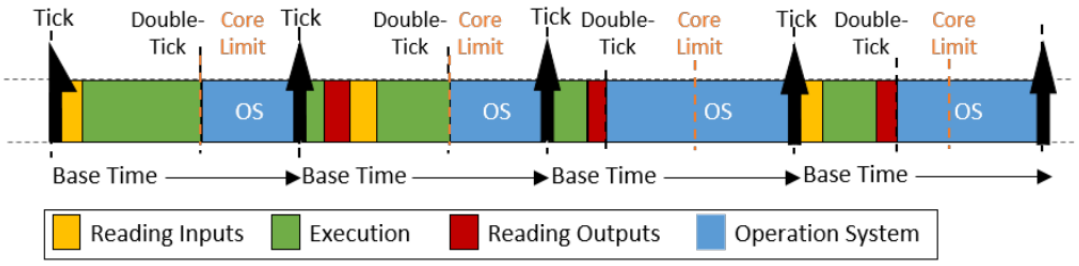
3.1.3 Çevrim süresinin aşıldığı durumlar

Bir görevin tanımlanan döngü süresi aşılırsa, bir önceki çevrimin kalan işlemleri sonraki döngüde devam eder. Ayrıca, döngü süresi aşma sayacı artırılır. Önceki döngünün işlenmesi tamamlandığında, sistem hemen geçerli döngünün görevlerini işlemeye başlar. Bu işlem mevcut döngü içinde tamamlanırsa, aşağıdaki şekilde gösterildiği gibi başka işlemler yapılır (Şekil 3.4).



Şekil 3.4 : Çevrim süresinde taşma olması durumu.

Doğrudan takip eden ikinci döngü de aşılırsa (bu durumda sistemin hala birinci döngüyü işlemesi veya ikinci devrenin başlayıp başlamadığı önemsizdir), geçerli işlemler tamamlanır; ancak sonraki görevin işlenmesi gerçekleşmez ve bir sonraki olası tike kadar bekler. Bu, birkaç çevrimin kaybedilebileceği anlamına gelir. Döngü süresi aşma sayacı buna göre artırılır (Şekil 3.5).



Şekil 3.5 : Çevrim süresinde üst üste iki kere taşma olması durumu.

3.2 EtherCAT

Yaklaşık 10 yıl önce piyasaya sürülmesinden bu yana, TwinCAT otomasyon yazılımı sürekli güçlendi ve on binlerce projede kullanıldı. Çoğu saha veri yolu ve genel iletişim ortamını düzgün ve optimize bir şekilde destekler. Klasik saha veri yollarının yanı sıra, farklı protokol çeşitlerinde ve farklı gerçek zamanlı davranışlarda USB ve Ethernet gibi normal bilgisayar ara yüzlerini de destekler. Ethernet söz konusu olduğunda,

TwinCAT uygulaması farklı Ethernet protokollerinin tam gerçek zamanlı bir şekilde çalıştırılmasını sağlar.

Açık, Ethernet tabanlı bir iletişim sistemi olan EtherCAT, otomasyon spektrumundaki çok çeşitli uygulamalar için tasarlanmıştır. Birkaç nanosaniye aralığında son derece yüksek verimlilik ve senkronizasyon kalitesi ile EtherCAT hızlı, yüksek performanslı makine kontrol sistemleri için idealdir.

TwinCAT ve EtherCAT kombinasyonu kullanıcılara yapılandırması ve teşhisi basit olan ve karmaşık görevler için bile kullanılabilecek çok uygun bir otomasyon sistemi sunar. Bu tür yapılandırma esnekliğinin gerekli olmadığı uygulamalarda, varsayılan ayarlar geleneksel saha veri yolu sistemlerinden çok daha iyi sonuçlar verir. Ayrıca, haberleşme sistemi manuel olarak yapılandırılabilir ve ilgili uygulamaya en uygun şekilde uyarlanır.

TwinCAT, bir EtherCAT sisteminin diğer tüm saha veri yolu sistemleri kadar kolay yapılandırılmasını ve yönetilmesini sağlar. Yeni bir master aygıt takılırsa veya otomatik olarak algılanırsa, diğer aygıtlar buna göre ayarlanır. TwinCAT Sistem Yöneticisi veri senkronizasyonu ve tutarlılığı ile ilgili tüm ayarları otomatik olarak hesaplar ve yönetir.

3.2.1 Çalışma prensibi

EtherCAT ile, herhangi bir bağlantı düğümündeki (node) standart Ethernet paketi, bir işlem verisi olarak alınamaz, yorumlanamaz ve kopyalanamaz. EtherCAT slave cihazlar, EtherCAT telegramları slave cihazdan geçerken kendilerine gönderilen verileri okur ve bu verileri anında işler. Telegramların slave cihazdan geçerken işlenmesi özelliğine “on the fly processing” denmektedir. Benzer şekilde, giriş verileri telgraf geçerken eklenir. Bir paket işlenmeden önce tam olarak alınmaz; bunun yerine işleme mümkün olan en kısa sürede başlar. Paketin gönderilmesi işlemi ayrıca küçük bit sürelerinin minimum gecikmesi ile gerçekleştirilir. Genellikle tüm ağ tek bir paket adreslenebilir.

3.2.2 Protokol

EtherCAT protokolü, işlem verileri için optimize edilmiştir ve Ethertype 0x88a4 kullanılarak doğrudan standart IEEE 802.3 Ethernet paketi içinde taşınır. Her biri 4 gigabayta kadar boyutta olabilen mantıksal proses imajlarının belirli bir bellek alanına

hizmet eden birkaç alt telgraftan oluşabilir. Veri dizisi, ağdaki düğümlerin fiziksel düzeninden bağımsızdır; adresleme herhangi bir sırada olabilir. Slave cihazlar arasında yayın, çok noktaya yayın ve haberleşme mümkündür, ancak master cihaz tarafından başlatılmalıdır. IP yönlendirmesi gerekiyorsa, EtherCAT protokolü UDP/IP datagramlarına eklenebilir. Bu ayrıca Ethernet protokol yığını ile herhangi bir kontrolün EtherCAT sistemlerini ele almasını sağlar.

3.2.3 Performans

Slave cihazlardaki ana mikroişlemciler, proses imajlarını aktarmak için Ethernet paketlerinin işlenmesine dahil olmadığından kısa döngü süreleri elde edilebilir. Tüm işlem veri iletişimi, slave kontrolör donanımında gerçekleştirilir. Fonksiyonel prensip ile birleştiğinde bu, EtherCAT'i yüksek performanslı dağıtılmış bir I/O sistemi haline getirir: 1000 dağıtılmış dijital I/O ile proses veri alışverişi yaklaşık 30 µs sürer, bu 100Mbit/s Ethernet üzerinden 125 bayt aktarımı için tipiktir. Tipik ağ güncelleme hızları 1–30 kHz'dir, ancak DMA yükü çok yüksekse EtherCAT daha yavaş döngü sürelerinde de kullanılabilir.

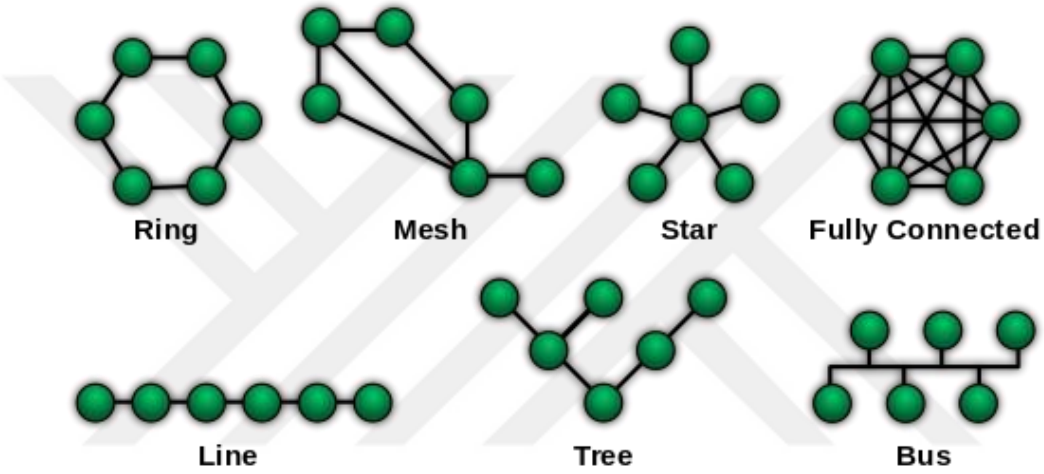
Her düğüm ve her veri için ayrı bir paket gerektirmediğinden, bant genişliği kullanımı en üst düzeye çıkarılır. Böylece, ≤ 100 µs olan çok kısa çevrim süreleri elde edilebilir. 100BASE-TX'in tam dubleks özelliklerini kullanarak 100 Mbit/s'den daha yüksek etkili veri hızlarına ulaşılabilir.

EtherCAT teknolojisi ilkesi ölçeklenebilir ve 100 Mbit/s değerine bağlı değildir. Gigabit Ethernet'e gelecekteki bir genişletme mümkündür, ancak şu anda hazırlık aşamasında değildir, çünkü EtherCAT performansı 100 Mbit/s'de yeterlidir.

3.2.4 Topoloji

Tam çift yönlü Ethernet fiziksel katmanları kullanarak, EtherCAT slave kontrolörleri açık bir bağlantı noktasını otomatik olarak kapatır ve başka bir EtherCAT slave algılanmazsa Ethernet paketini döndürür. Slave aygıtlar bir, iki veya daha fazla bağlantı noktasına sahip olabilir. Bu özellikler nedeniyle EtherCAT, çizgi, ağaç, halka, yıldız veya bunların herhangi bir kombinasyonu dahil olmak üzere birçok sayıda ağ topolojisi destekler (Şekil 3.6). Protokol ayrıca kablo yedekli çalışma, “Hot Connect”, çalışma sırasında cihazların değiştirilmesi vb. özellikler sağlar.

Tutarlı senkronizasyona sahip sıralı veya komşu kontrol sistemleri gibi topoloji varyasyonlarının ve farklı ağ mimarileri sayısız olasılık sağlar. Ek anahtarları gerekli değildir. Ethernet fiziği, iki düğüm arasında 100 m'ye kadar kablo uzunluğuna izin verir, bu nedenle E-bus sadece modüler cihazlar için fiziksel katman olarak kullanılmak üzere tasarlanmıştır. Her kablo yolu için, sinyal değişkeni ayrı ayrı seçilebilir. Daha yüksek mesafeler veya iki slave arasında tam galvanik izolasyon için fiber optik kablolar kullanılır. Tek modlu fiber ile, iki düğüm arasındaki 20 km'ye kadar mesafeler köprülenebilir. Ağ parçası başına toplam 65535 düğüm bağlanabildiğinden, ağ genişlemesi neredeyse sınırsızdır.



Şekil 3.6 : Topoloji çeşitleri.

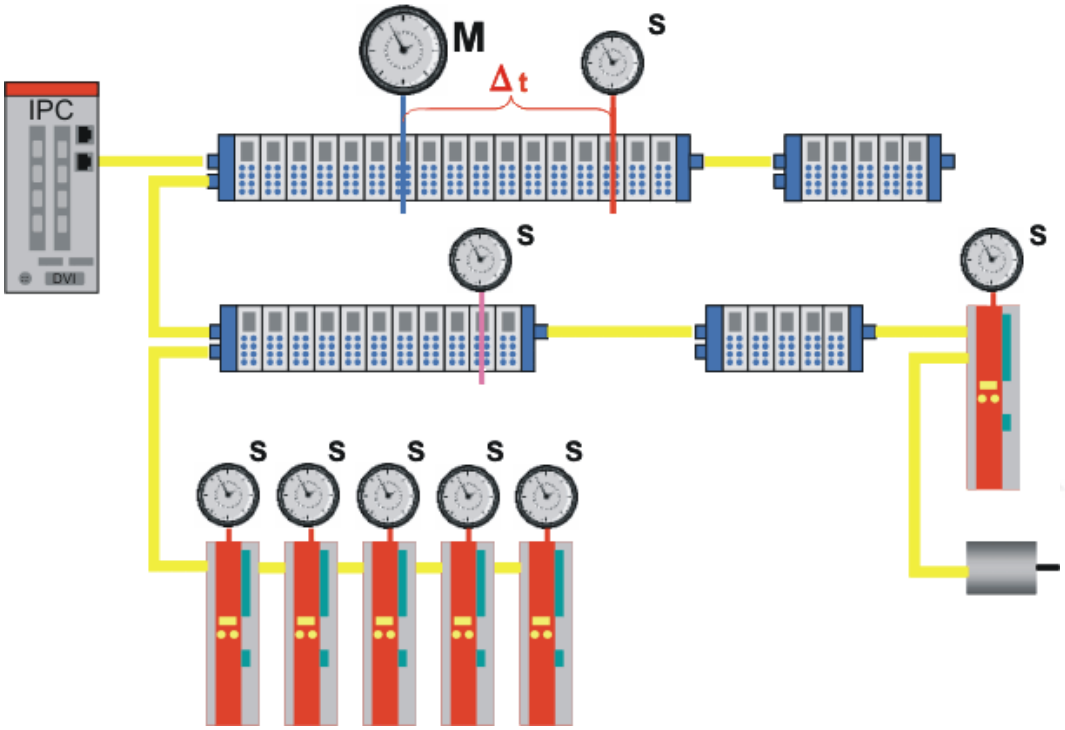
3.2.5 Senkronizasyon

Senkronizasyon için çok düşük kıpırdamaya yol açan dağıtılmış bir saat mekanizması uygulanır (Şekil 3.7). Bu nedenle, EtherCAT ana (master) cihazda özel bir donanım gerektirmez ve özel haberleşme yardımcı işlemcisi olmasa bile herhangi bir standart Ethernet MAC üzerindeki yazılıma uygulanabilir.

Dağıtılmış bir saatin kurulmasına yönelik tipik süreç, ana cihaz tarafından tüm yardımcı cihazlara (slave) belirli bir adrese bir yayın gönderilerek başlatılır. Bu mesajın alınması üzerine, tüm yardımcılar dahili saatlerinin değerini iki kez, bir kez mesaj alındığında ve bir kez geri döndüğünde mandallar (Latch). Master daha sonra tüm mandallı değerleri okuyabilir ve her ikincil cihaz için gecikmeyi hesaplayabilir. Bu işlem, kıpırdama ve ortalama çıkış değerlerini azaltmak için gerektiği kadar tekrar edilebilir. Toplam gecikmeler, slave halkasındaki konumlarına bağlı olarak her slave için hesaplanır ve bir ofset kaydına yüklenir. Son olarak, ana cihazın sistem saatinde

bir yayın okuma-yazma yayınlar, bu da ilk dağıtılmış saatli yardımcı cihazı referans saati yapar ve diğer tüm yardımcı cihazlar dahili saatlerini şimdi bilinen ofsetle uygun şekilde ayarlamaya zorlar.

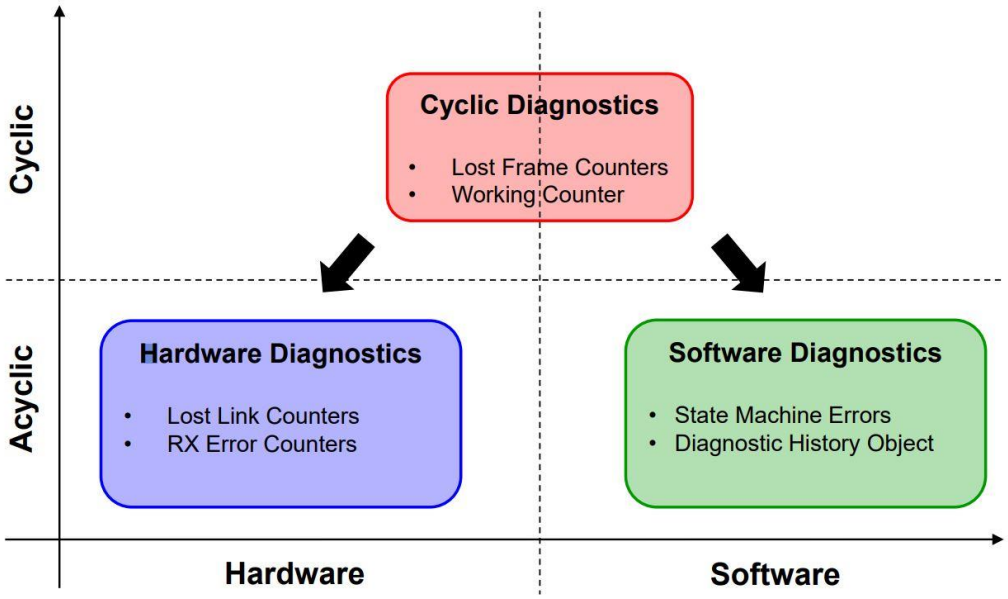
Başlatmadan sonra saatleri senkronize tutmak için, ana veya yardımcı cihazlar, her yardımcı cihazın dahili saatleri arasındaki hız farkının etkilerine karşı koymak için yayını düzenli olarak tekrar göndermelidir. Her yardımcı cihaz, iç saatlerinin hızını ayarlamalı veya her ayarlaması gerektiğinde bir iç düzeltme mekanizması uygulamalıdır.



Şekil 3.7 : EtherCAT sisteminde dağıtılmış saat.

3.2.6 Teşhis

Sorunların hızlı ve hassas tespiti EtherCAT'in birçok teşhis özelliğinden biridir. İletim sırasındaki bit hataları, CRC kontrol toplamının analizi ile güvenilir bir şekilde tespit edilir: 32 bit CRC polinomunun minimum “Hamming” mesafesi 4'tür. Hata algılama ve yerelleştirme protokolünün yanı sıra, EtherCAT sisteminin iletim fiziği ve topolojisi tüm iletim yollarının kalitesinin izlenmesine olanak sağlar. İlgili hata sayaçlarının otomatik analizi, kritik ağ parçalarının tam olarak yerelleştirilmesini sağlar (Şekil 3.8).



Şekil 3.8 : EtherCAT teşhis yöntemleri.

3.2.7 İzleme

EtherCAT, IEEE 802.3'e göre standart Ethernet paketleri kullandığından, EtherCAT iletişimini izlemek için herhangi bir standart Ethernet aracı kullanılabilir. Ek olarak, kaydedilen EtherCAT veri trafiğinin rahatlıkla hazırlanabileceği ve görüntülenebileceği "Wireshark" veya Microsoft ağ monitörü için ücretsiz ayrıştırıcı yazılımı vardır.

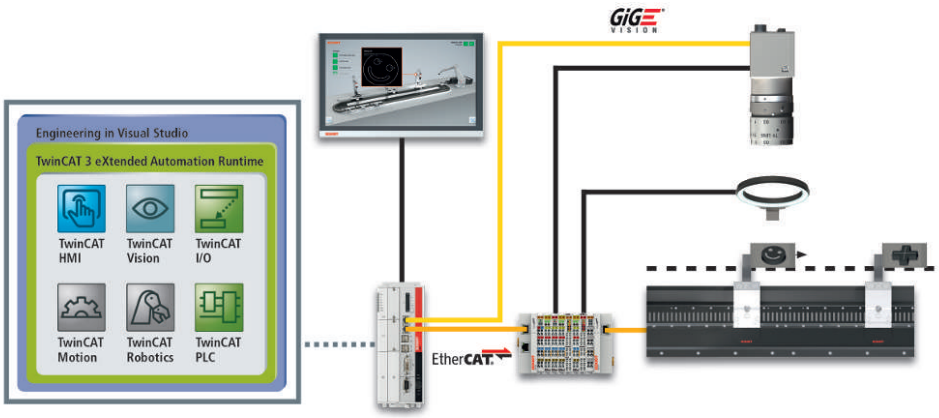
3.2.8 Ağ geçitleri

Ağ geçitleri kullanılarak CANopen, DeviceNet veya PROFIBUS gibi mevcut ağlar EtherCAT ortamına sorunsuz bir şekilde entegre edilebilir. Ayrıca, ağ geçitleri geleneksel bir saha veri yolundan EtherCAT'a sorunsuz bir geçiş yolu sunarak yatırım maliyetini düşürür.

EtherCAT'in performansı sayesinde, harici saha veri yolu ana cihazlarla iletişim, PCI veya diğer temel veri yolları ile bağlanan geleneksel kartlar kadar hızlıdır. Merkezi olmayan saha veri yolu arabirimleri daha kısa uzantılara yol açtığından, geleneksel mimaride mümkün olandan daha yüksek baud hızlarıyla çalıştırılabilirler.

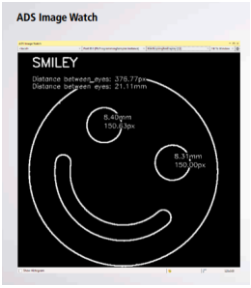
3.3 TwinCAT Vision

TwinCAT Vision, TwinCAT 3'ün bir modülüdür ve endüstriyel görüntü işleme uygulamalarında kullanılmaktadır (Şekil 3.9). Objelerin tanınması, algılanması veya ölçülmesi gibi işlemler, PLC içinde gerçek zamanlı bir şekilde yapılır. Görüntü işlemenin TwinCAT ile entegrasyonu sayesinde, yüksek senkronizasyon ve aşırı kısa tepki süreleri elde edilebilir.



Şekil 3.9 : TwinCAT Vision'a genel bakış.

"Tc3_Vision" isimli PLC kütüphanesini ile birlikte, IEC61131-3 dillerinden herhangi birini kullanarak görüntü işleme sekansları gerçekleştirilebilir. Görüntü işlemenin adımları, "ADS Image Watch" ile herhangi bir adımda görüntülenebilir (Şekil 3.10).



Şekil 3.10 : ADS imaj görüntüleme aygıtı.

TwinCAT Vision gerçek zamanlı çalışmaya uygundur çünkü bütün görüntü işleme algoritmaları senkron bir şekilde kontrolcünün içindeki TwinCAT 3 ortamında çalışmaktadır ve çok çekirdekli çalışmayı desteklemektedir.

TwinCAT Vision GigE Vision standardını desteklemektedir. Bu sayede birçok markanın birçok modeli sorunsuz bir şekilde bağlanabilir. Kamera konfigürasyonu, kalibrasyonu vb. gibi işlemler için TwinCAT mühendislik ortamı kullanılabilir.

3.3.1 Gigabit Ethernet (GigE)

Gigabit Ethernet (GigE), endüstriyel görüntü işleme alanındaki dijital kameralar için en hızlı büyüyen arabirimdir. Hemen hemen her uygulamada analog cihazların yerini alabilecek kameralar üretme potansiyeli sağlayan, evrensel olarak uygulanabilir bir dijital arayüzdür.

Gigabit Ethernet, bant genişliği, kablo uzunluğu ve çoklu kamera işlevselliği açısından en geniş teknolojik esnekliği sunar. 100 MB/s'ye kadar veri hızları ve maksimum 100 metre kablo uzunluğu için doğru seçimdir. Birkaç kameranın olduğu durumlarda bile karmaşık kurulumları çok basit hale getirir.

3.3.2 GigE vision standardı

GigE Vision ise Gigabit Ethernet haberleşme protokolünün kullanan, global bir kamera arayüz standardıdır (Şekil 3.11). GigE Vision ile GigE kesinlikle karıştırılmamalıdır ve kamera seçimi yapılırken GigE Vision destekleyen endüstriyel kameralar seçilmelidir, sadece GigE destekleyenler değil çünkü her GigE destekleyen endüstriyel kamera, GigE Vision standardını desteklemek zorunda değildir.



Şekil 3.11 : GigE Vision.

GigE Vision'ın aşağıdaki gibi avantajları vardır.

- Hızlıdır ve 125 MB/s gibi yüksek bant genişliğini desteklemektedir. Bu sayede yüksek çözünürlüklü yani yüksek boyutlu imajlar çok kısa bir sürede aktarılabilir.
- Gigabit Ethernet kablosu kullanıldığı için en fazla 100 metre uzaktaki bir kamera ile haberleşebilir.
- Düşük maliyetli CAT5 veya CAT6 kabloları ve standart konektörler kullanılabilir.
- Ethernet'in gelişimi ile birlikte yüksek oranda ölçeklendirilebilir.
- Kamera bağlantısı için az ve standart ekipmanlar gerektirdiği için ekonomiktir.

Gigabit Ethernet 1 Gbit/s veri aktarma hızına sahiptir. Bir imajı ne kadar hızlı aktardığını anlamak için aşağıdaki sayısal örnek irdelenebilir.

$$1\ Gbit/s\ = 125\ MB/s \tag{3.1}$$

$$Aktarım\ süresi = \frac{İmaj\ boyutu}{Aktarım\ hızı} \tag{3.2}$$

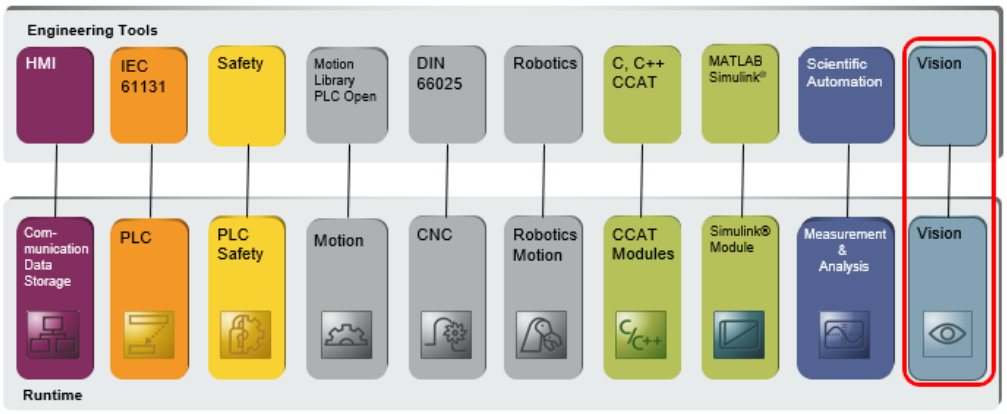
Elimizdeki 1.25 MB boyutundaki bir imaj dosyasının aktarım süresi ne kadardır? Formül 3.1 ve 3.2 kullanılarak denklem 3.3 oluşturulabilir.

$$Aktarım\ süresi = \frac{1.25\ MB}{1Gbit/s} = \frac{1.25\ MB}{125\ MB/s} = 0.01s = 10ms \tag{3.3}$$

Denklem 3.3'ün sonucundan da görülebileceği üzere, 1.25 MB boyutundaki bir imaj dosyası 10 milisaniyede Gigabit Ethernet üzerinden aktarılabilir.

3.3.3 TwinCAT Vision bileşenleri

Diğer TwinCAT modülleri (Şekil 3.12) gibi, TwinCAT Vision'da Visual Studio tabanlı olan TwinCAT mühendislik ortamına tamamen entegredir.



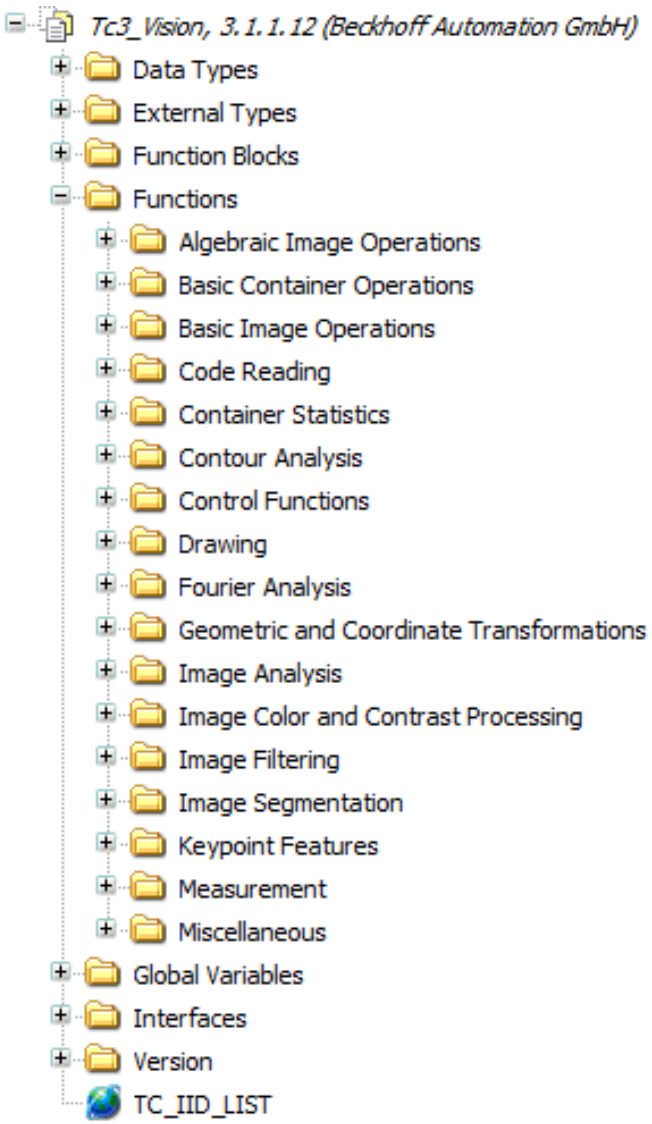
Şekil 3.12 : TwinCAT 3'ün modüler yapısı ve bazı modülleri.

3.3.3.1 Konfigürasyon asistanları

Endüstriyel kameraları bağlamak, kalibre etmek, konfigüre etmek ve simülasyonunu yapmak için asistanlar mevcuttur ve bu asistanlara TwinCAT 3 mühendislik ortamından erişilebilir. Kameranın olmadığı durumlarda harici imajlar ile de çevrimdışı bir şekilde çalışılabilir.

3.3.3.2 PLC kütüphanesi

TwinCAT Vision görüntü işlemeye ait çok sayıda algoritmayı PLC kütüphanesi olaraktan kullanıcıya sunmaktadır. Kamera arayüzleri ve çevrimdışı dosyalar ile çalışmaya yarayan fonksiyon blokları da içermektedir. Kütüphane içeriği (Şekil 3.13) TwinCAT projesinin referanslar sekmesinden incelenebilir.



Şekil 3.13 : TwinCAT Vision PLC kütüphane içeriği.

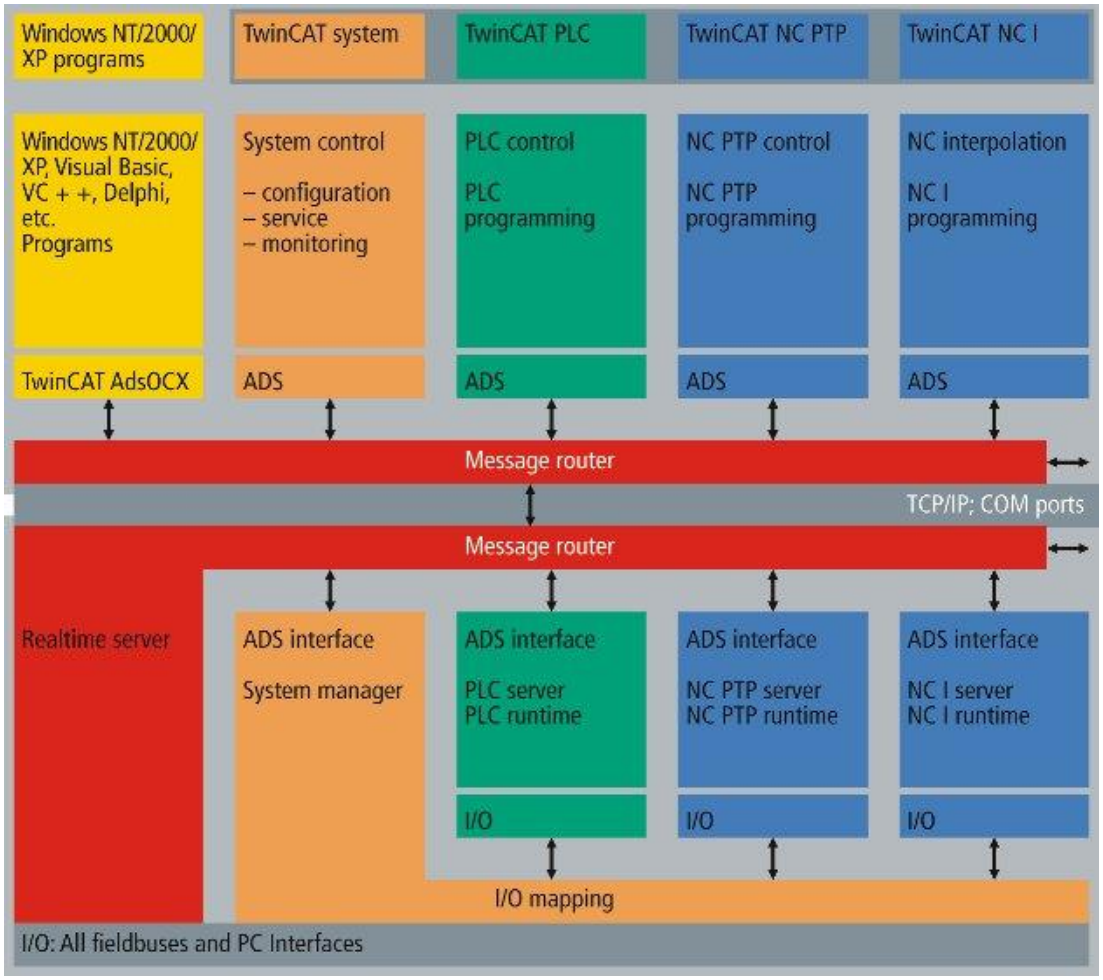
3.3.3.3 GigE Vision sürücüsü

TwinCAT Vision, gerçek zamanlı yetenekli bir GigE Vision sürücüsü içerir. Kamera yapılandırması için geliştirme ortamında ve TwinCAT çalışma zamanında kullanılır. GigE Vision standardı UDP/IP üzerinden endüstriyel kameralarla iletişim kurar. Bu

standart ağdaki kameraları bulmak, yapılandırmak ve kameralardan görüntü verileri almak için kullanılır.

3.4 Automatic Device Specification (ADS)

ADS teknolojisi hem dahili hem de harici olarak kullanılabilen bir protokol olarak düşünülebilir. TwinCAT kurulu olan her bilgisayar ADS haberleşmesi yapabilir hale gelmektedir. ADS konsepti (Şekil 3.14) aşağıda gösterilmiştir.



Şekil 3.14 : ADS konsepti.

Bir ADS cihazının kendine has olmasını sağlayan iki faktör vardır. Bunlardan bir tanesi ADS portu, diğer ise ADS adresi ya da diğer adıyla AMS adresidir.

TwinCAT sistem mimarisi, yazılımın münferit modüllerinin (i.e. TwinCAT PLC, TwinCAT NC, ...) bağımsız cihazlar olarak işlemesine izin verir: Her görev için bir yazılım modülü ("Sunucu" veya "İstemci") vardır.

Bu nesneler arasındaki mesajlar, mesaj yönlendirici (message router) tarafından tutarlı bir ADS arayüzü üzerinden değiştirilir. Bu, sistemdeki ve TCP/IP bağlantıları üzerindeki tüm iletileri yönetir ve dağıtır. Mesaj yönlendiricileri her TwinCAT kurulu bilgisayarda bulunur.

Bu, tüm TwinCAT sunucusu ve istemci programlarının komut ve veri alışverişi yapmasına izin verir. ADS arabirimini uygulayan (böylece ADS üzerinden erişilebilen) ve "sunucu hizmetleri" sunan bir nesne ADS aygıtı olarak bilinir. ADS hizmetinin ayrıntılı anlamı, her ADS aygıtına özgüdür.

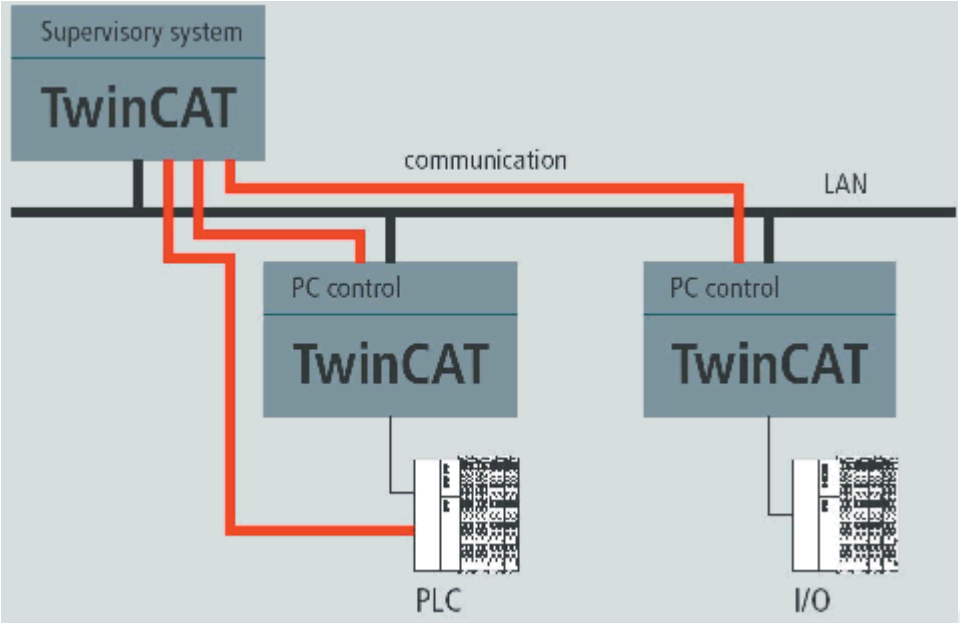
Çizelge 3.1 : Bilinen ADS portları ve açıklamaları.

ADS portu	ADS cihaz açıklaması
100	Logger
110	Eventlogger
300	IO
301	Ek görev 1
302	Ek görev 2
500	NC
801	PLC görevi 1
811	PLC görevi 2
821	PLC görevi 3
831	PLC görevi 4
900	CAM kontrolcüsü
10000	Sistem servisi
14000	Ölçüm servisi

Çizelge 3.1'de görüldüğü gibi, önceden belirli bazı ADS portları vardır. Bu ADS portları kullanılarak veri okuma mümkün olacağı gibi, bu portlara veri yazılması da mümkündür. Örneğin bir C# uygulaması ile Beckhoff PLC'ye bağlanmış olan saha aygıtlarına 300 portundan erişip, gerekli kontroller yapılabilir ya da 801 portundan PLC değişkenlerine kolaylıkla erişilebilir.

Bir önceki paragrafta belirtildiği gibi, ADS kütüphanesi C++, C#, Java gibi birçok yüksek seviye programlama dilleri için mevcuttur. Bu uygulamada ise Python programlama dili kullanılıp, C6030 ile haberleşme sağlanacaktır.

ADS protokolü çoklu haberleşme modelini (Şekil 3.15) desteklemektedir. Yani birden çok istemci olduğu durumlarda, sunucu tüm istemcilere cevap verebilmektedir.



Şekil 3.15 : Çoklu haberleşme modeli.

3.5 TwinCAT HMI

TwinCAT HMI, insan-makine arayüzleri oluşturmak için kullanılan bir yazılımdır. TwinCAT gibi, doğrudan Visual Studio'ya entegre edilmiştir. Konfigürasyon için bir grafik WYSIWYG editörü kullanılır, yani programlama gerektirmez.

Araç kutusundan gelen kontroller kullanıcı arayüzünde düzenlenebilir ve gerçek zamanlı değişkenlerle bağlanabilir. Kullanıcı kontrolleri, belirli kontrolleri grafiksel olarak oluşturmak ve yapılandırmak için kullanılabilir. JavaScript geliştiricileri kendi objelerini geliştirebilir. Geliştirme sırasında, işlevsellik gerçek zamanlı sistemden canlı verilerle test edilebilir. Bu canlı modda değişiklikler yapmak da mümkündür.

HMI mantığı, JavaScript'te istemci tarafında veya bir sunucu uzantısı olarak uygulanabilir. Bir sunucu uzantısı ise C # 'da uzantılar geliştirmenizi sağlar.

HMI web tabanlı ve çözünürlük uyumlu olduğundan çözünürlük ve yönelim otomatik olarak ayarlanır. Bu, aynı sayfanın farklı ekran boyutları, en boy oranları ve yönler için kullanılmasına izin verir. Şekil 3.16'da farklı ekran boyutlarında ve farklı ekran yönlerinde TwinCAT HMI gösterilmiştir. HMI'ın web tabanlı olması demek aslında oluşturulan HMI tasarımındaki her bir objenin arka planda bir HTML5 kodu olduğudur. Bu özelliğinden dolayı HTML5 destekleyen bütün modern tarayıcılarda sorunsuz bir şekilde açılmaktadır. Bu özelliğinden dolayı da birçok senaryoda kullanılabilir.



Şekil 3.16 : Farklı ekran boyutlarında TwinCAT HMI.

3.5.1 Kullanım senaryoları

HMI çok çeşitli senaryolara entegre edilebilir.

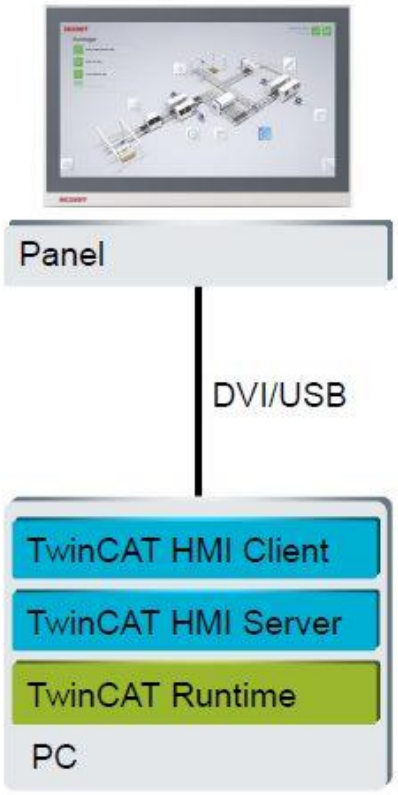
Bir ağda, farklı denetleyiciler verilerini bir veya daha fazla istemciyle değiştirebilir. İstemciler yerel PC'lerde veya Panel PC'lerde veya tablet veya akıllı telefon gibi mobil cihazlarda çalışabilir.

3.5.1.1 Yerel istemci

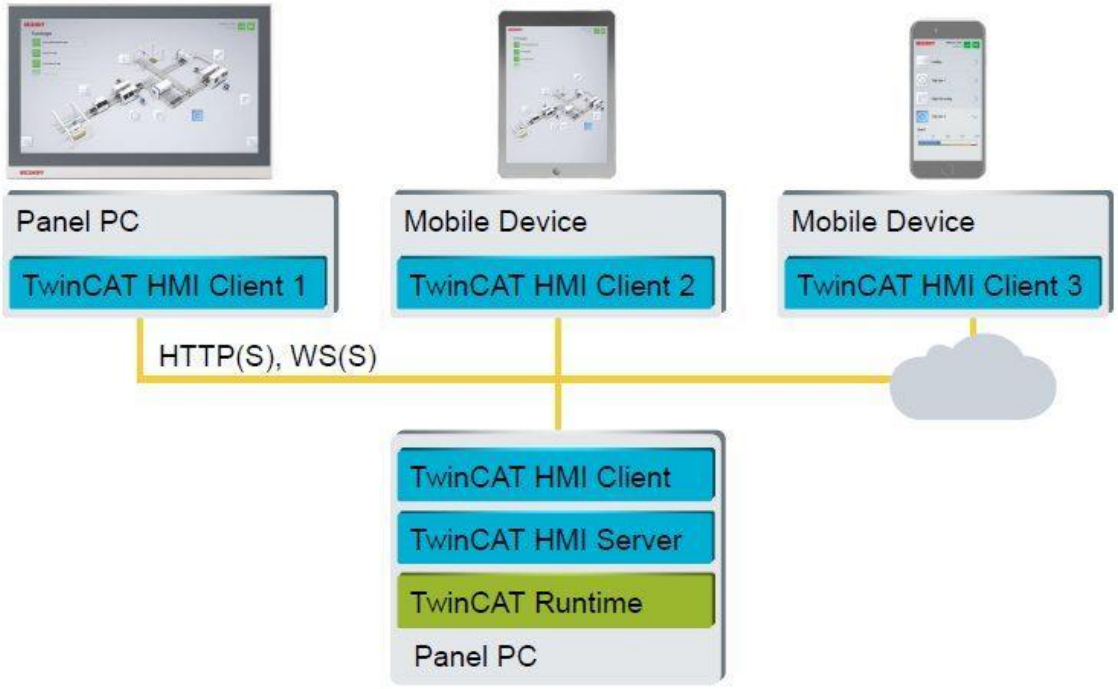
Yerel istemci senaryosunda bir tane istemci ve bir tane de sunucu bulunmaktadır. Bu tip senaryo web tabanlı olmayan HMI'larda kullanılan senaryodur ve endüstride en çok karşılaşılan modeldir. Bu model Şekil 3.17'de gösterilmiştir ve endüstriyel bilgisayara bağlı bir panelde HMI gösterilebilir.

3.5.1.2 Çoklu istemci

Bir HMI farklı çözünürlükler için yapılandırılabilir ve aynı anda birkaç cihaz tarafından çağrılabilir. Bu senaryo ise Şekil 3.18'de gösterilmiştir. Bağlanacak istemci sayısı lisans ile sınırlıdır ve arttırılabilir.



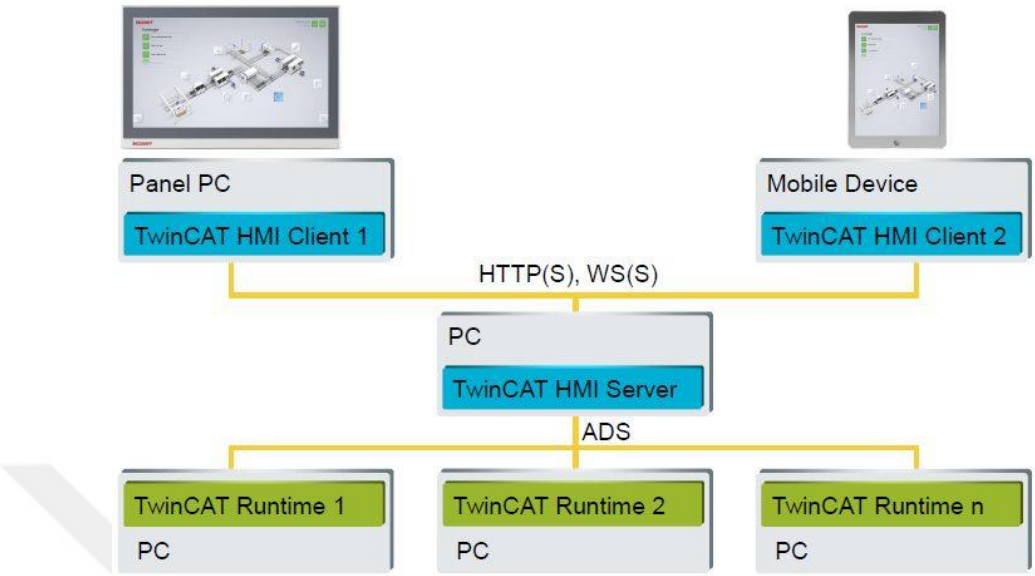
Şekil 3.17 : Yerel istemci senaryosu.



Şekil 3.18 : Çoklu istemci senaryosu.

3.5.1.3 Çoklu çalışma zamanı

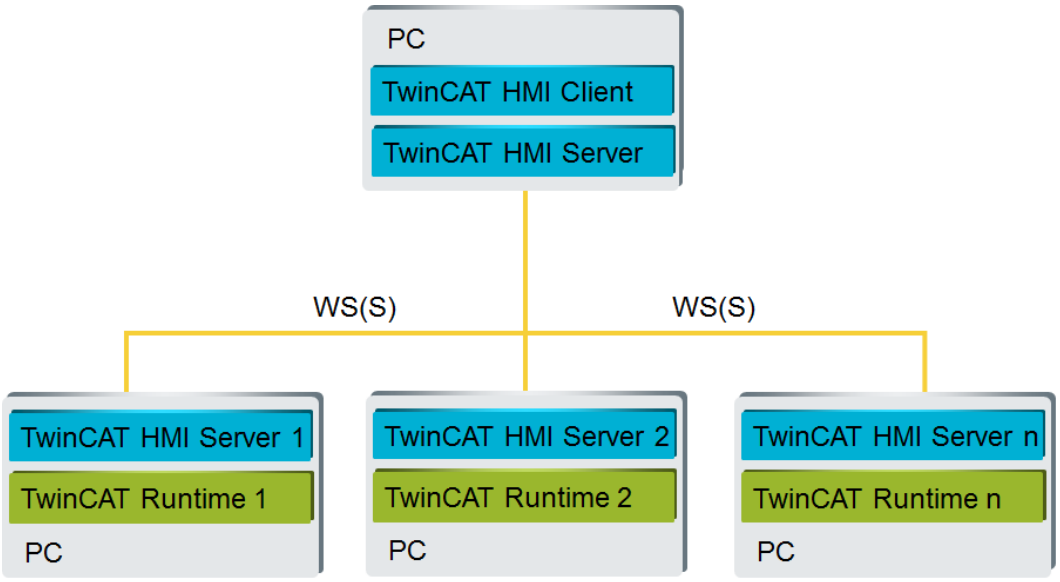
TwinCAT HMI birçok gerçek zamanlı sistemi bağlayabilir (Şekil 3.19).



Şekil 3.19 : Çoklu çalışma zamanı senaryosu.

3.5.1.4 Çoklu sunucu

TwinCAT HMI sunucusu diğer TwinCAT HMI sunucularıyla haberleşebilir (Şekil 3.20). Bu, sunucudan sunucuya haberleşen yapılar kurmaya olanak sağlar. Sunucudan sunucuya haberleşme, sunucular arasında herhangi bir veri noktası alışverişi yapma imkanı sunar. Bu, daha yüksek seviyeli sistemlerin oluşturulmasını sağlar.



Şekil 3.20 : Çoklu çalışma zamanı senaryosu.

3.6 Python

Python yaygın olarak kullanılan genel amaçlı, üst düzey bir programlama dilidir. 1991 yılında Guido van Rossum tarafından oluşturuldu ve Python Yazılım Vakfı tarafından daha da geliştirildi. Kod okunabilirliğine vurgu yapılarak tasarlandı ve sözdizimi, programcıların kavramlarını daha az kod satırında ifade etmelerini sağlar.

Python, hızlı çalışmanızı ve sistemleri daha verimli bir şekilde entegre etmenizi sağlayan bir programlama dilidir.

İki ana Python sürümü vardır: Python 2 ve Python 3. Her ikisi de oldukça farklıdır. Bu uygulama da Python 3 kullanılacaktır. TwinCAT ile haberleşmek için PyAds isimli kütüphane kullanılırken, yapay sinir ağları için de Tensorflow kullanılacaktır.

3.6.1 Tensorflow

TensorFlow, makine öğrenimi için uçtan uca açık kaynaklı bir platformdur. Araştırmacıların makine öğrenmesinde en son teknolojiyi kullanmasına olanak tanıyan kapsamlı ve esnek bir araçtır.

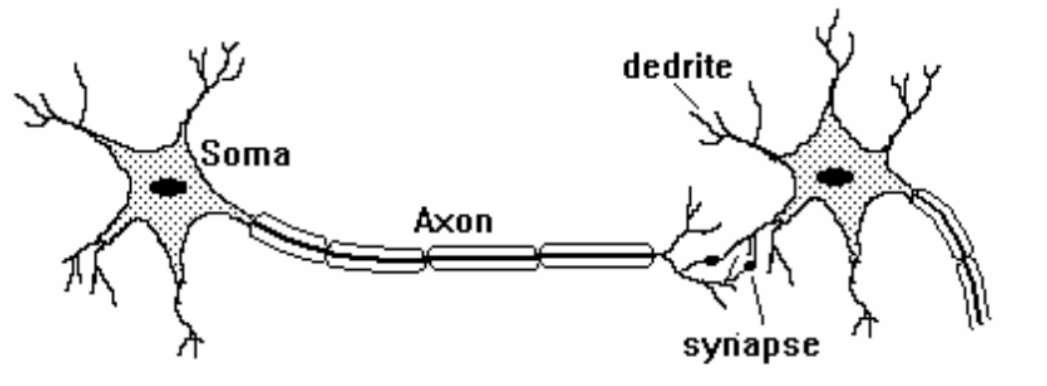
3.6.2 PyAds

Bu, TwinCAT'in ADS kütüphanesi için bir Python paketidir. TwinCAT cihazlarıyla iletişim kurmak için python fonksiyonları sağlar.

4. YAPAY SİNİR AĞLARI

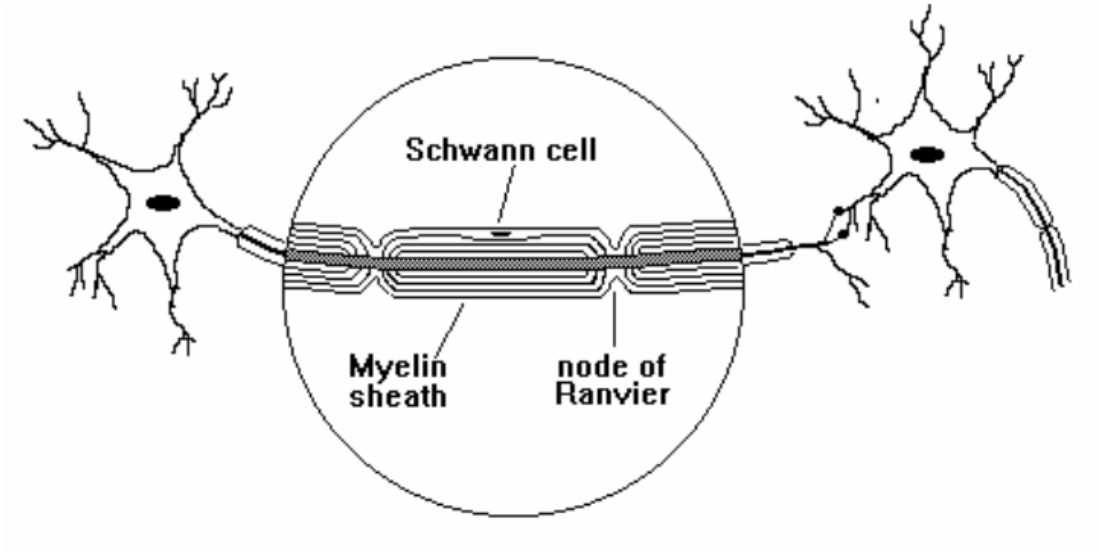
İnsan merkezi sinir sisteminin yaklaşık $1.3x10^{10}$ nörondan oluştuğu ve bunların yaklaşık $1x10^{10}$'unun beyinde olduğu iddia edilmektedir. Herhangi bir zamanda, bu nöronların bazıları ateşlenir ve bu elektriksel aktiviteye bağlı güç kaybının 10 W olduğu tahmin edilmektedir. Beyindeki aktivitenin izlenmesi, uykuda bile saniyede $5x10^{7}$ sinir darbesinin beyin ve vücudun diğer kısımları arasında ileri geri aktarıldığını göstermiştir. Bu oran uyanıkken önemli ölçüde artmaktadır [11].

Bir nöronun soma adı verilen kabaca küresel bir hücre gövdesi vardır (Şekil 4.1). Somada üretilen sinyaller, hücre gövdesi üzerinde akson veya sinir lifleri adı verilen bir uzantı yoluyla diğer nöronlara iletilir. Hücre gövdesi etrafındaki gür ağaç gibi bir başka uzantı türü, diğer nöronlar tarafından üretilen sinyalleri almaktan sorumlu olan dendritlerdir [12].



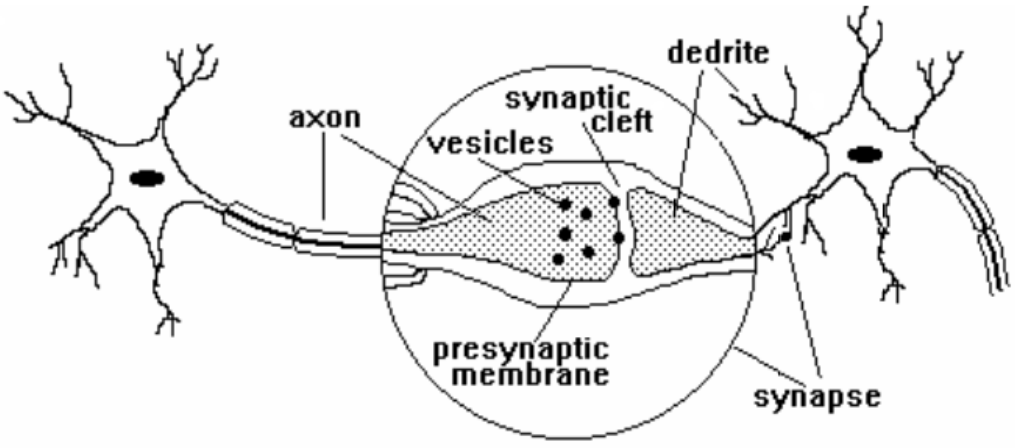
Şekil 4.1 : Biyolojik nöron.

İnsan vücudundaki bir milimetreden bir metreye kadar değişen bir uzunluğa sahip bir akson (Şekil 4.2), akson yükseltisi adı verilen noktada hücre gövdesinden uzanır. Diğer uçta, akson birkaç uca ayrılır, en ucunda akson büyür ve terminal düğmeleri oluşturur. Terminal düğmeleri, sinyallerin bir nörondan diğerine ilettiği kavşaklar olan sinaps adı verilen özel yapılara yerleştirilir (Şekil 4.3). Bir nöron tipik olarak 10^{3} ila 10^{4} sinaptik kavşak kullanır.



Şekil 4.2 : Akson.

Binlerce kimyasal ileticinin molekülünü tutan sinaptik veziküller terminal düğmelerinde yer alır. Sinapsa bir sinir impulsu geldiğinde, bu kimyasal vericilerin bazıları sinaptik boşluk içine boşaltılır; bu sinyal ileten nöronun terminal düğmesi ile onu alan nöronun zarı arasındaki dar boşluktur. Genelde sinapslar bir nöronun bir akson dalı ile bir diğerinin dendriti arasında gerçekleşir. Çok yaygın olmasa da sinapslar iki akson veya farklı hücrelerin iki dendriti arasında veya bir akson ile bir hücre gövdesi arasında da meydana gelebilir.

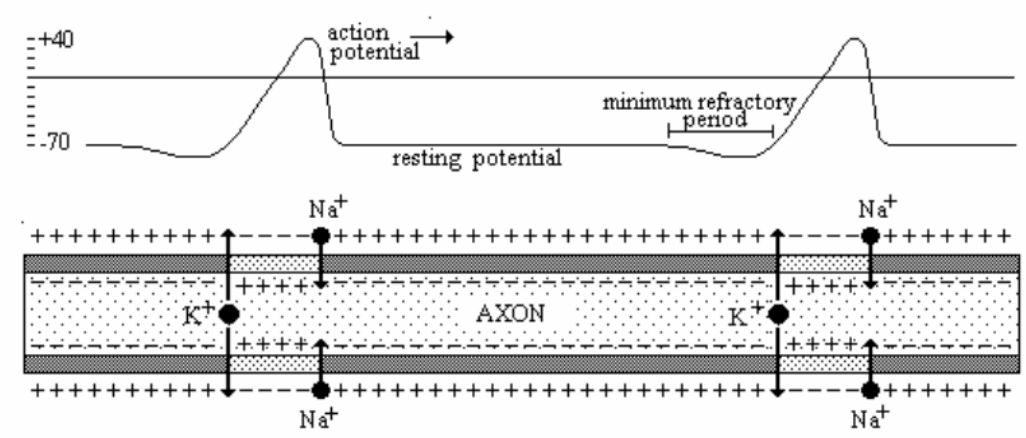


Şekil 4.3 : Sinaps.

Nöronlar, sadece 5 nanometre kalınlığında yarı geçirgen bir zar ile kaplıdır. Zar hücre içi sıvıdaki iyonları seçici olarak emebilir ve reddedebilir. Zar temel olarak hücre içi sıvı ile hücre dışı sıvı arasında farklı bir iyon konsantrasyonu sağlamak için bir iyon

pompası görevi görür. Sodyum iyonları hücre içi sıvıdan hücre dışı sıvıya sürekli olarak çıkarılırken, bir denge durumunu korumak için potasyum iyonları hücre dışı sıvıdan emilir. İç ve dış iyon konsantrasyonlarındaki farktan dolayı hücre zarı polarize olur. Dengede hücrenin iç kısmının, hücrenin dışına göre 70 mV negatif olduğu görülür. Sözü edilen potansiyele dinlenme potansiyeli denir.

Bir nöron sinaptik bağlantıları aracılığıyla çok sayıda nörondan girdi alır. Presinaptik hücre zarına gelen sinir sinyalleri, kimyasal vericilerin sinaptik yarık içine salınmasına neden olur. Bu kimyasal vericiler boşluk boyunca yayılır ve reseptör bölgesinin postsinaptik zarına katılır. Postsinaptik hücrenin zarı kimyasal vericileri toplar. Bu, sinaptik yarıkta salınan kimyasalların türüne bağlı olarak, kademeli potansiyel olarak adlandırılan soma potasyonunda bir azalmaya veya artışa neden olur. Depolarizasyonu teşvik eden sinaps türlerine uyarıcı denir ve cesaretini kırıcı olanlara inhibitör sinapslar denir. Polarizasyondaki azalma bir eşiği aşmak için yeterliyse, post-sinaptik nöron ateşlenir (Şekil 4.4).



Şekil 4.4 : Aksondaki aksiyon potansiyeli.

İmpulsların uyarıcı sinapslara gelişi, somanın depolarizasyonuna katkıda bulunurken, engelleyici etki, uyarıcı dürtünün depolarize edici etkisini iptal etme eğilimindedir. Genel olarak, tek bir sinapstan kaynaklanan depolarizasyon nöronu ateşlemek için yeterli olmasa da zarın diğer alanları sinir uyarılarının diğer sinapslardan gelmesiyle aynı zamanda depolarize edilirse, eşiği aşmak yeterli olabilir ve ateşleme gerçekleşir.

Akson yükseltisinde, uyarıcı etkiler hücre zarından düzenli iyon aktarımının kesintiye uğramasıyla sonuçlanır, böylece iyonlar zardan yayıldıkça iyonik konsantrasyonlar hemen eşitlenmeye başlar. Depolarizasyon yeterince büyükse, zar potansiyeli

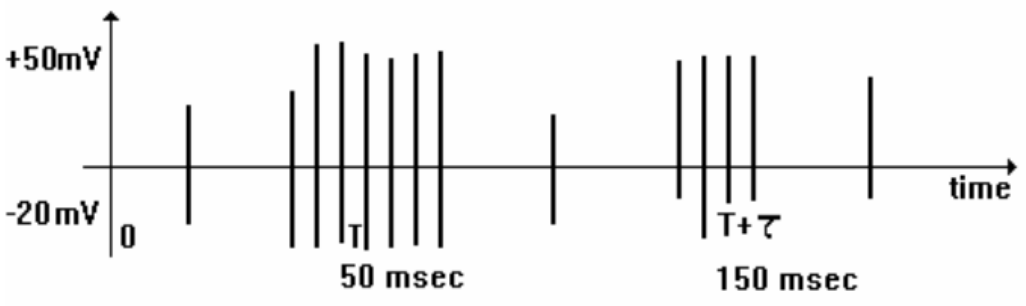
nihayetinde çöker ve kısa bir süre için iç potansiyel pozitif olur. Aksiyon potansiyeli, potansiyeldeki bu kısa tersinimin adıdır, bu da hareket potansiyeli olan bölgeden hareket potansiyeli olan akson üzerindeki bitişik bir bölgeye akan bir elektrik akımıyla sonuçlanır. Bu akım, bir sonraki dinlenme bölgesinin potansiyelinin değişmesine neden olur, bu nedenle etki, akson zar boyunca bu şekilde yayılır.

Bir aksiyon potansiyeli belirli bir noktayı geçtikten sonra, nöron bir süre tekrar uyarılamaz ve buna da refrakter dönem denir. Nöronun depolarize edilmiş kısımları iyileşme durumunda olduğundan ve hemen tekrar aktif olamadığından, elektriksel aktivitenin nabzı her zaman sadece ileri yönde yayılır. Akson üzerinde daha önce tetiklenen bölge daha sonra sodyum potasyum pompalarının etkisinden dolayı hızla polarize dinlenme durumuna geri döner. Ateşe dayanıklılık süresi yaklaşık 1 milisaniyedir ve bu sinir darbesi iletimini sınırlar, böylece bir nöron tipik olarak saniyede 1000 impulsa kadar bir hızda ateşlenebilir ve sinir impulsları üretebilir. İmpuls sayısı ve belirli bir nörona sinaptik kavşaklara ulaşma hızı, toplam uyarıcı depolarizasyonun, nöronun ateşlenmesine neden olmak ve böylece aksonuna bir sinir impulsu göndermek için yeterli olup olmadığını belirler. Depolarizasyon etkisi soma zarı boyunca yayılabilir, ancak bu etkiler akson yükseltisine ulaşmadan dağılabilir. Bununla birlikte, sinir impuls akson yükseltisine ulaştığında, depolarizasyon etkisinin kimyasal vericilerin sinaptik yarık içine salınmasına neden olacağı sinapslara ulaşana kadar yayılır.

Aksonlar genellikle aksonun büyümesini destekleyen birçok Schwann hücresi katmanından oluşan miyelin kılıfı ile çevrelenir. Aksonun aşağıya yayılma hızı, aksonun hücre dışı sıvıdan yalıtılmasını sağlayan ve iyonların zar boyunca iletimini önleyen miyelin kılıfının kalınlığına bağlıdır. Miyelin kılıfı, hücre dışı sıvının zarla temas ettiği ve iyonların transferinin gerçekleştiği Ranvier düğümleri adı verilen dar ve düzenli aralıklarla kesilir. Aksonların kendileri zayıf iletkenler olduğundan, Ranvier düğümlerinde depolarizasyonlar meydana geldikçe aksiyon potansiyeli iletilir. Bu, ardışık bir şekilde olur, böylece bir düğümün depolarizasyonu bir sonrakinin depolarizasyonunu tetikler. Sinir impulsu, kayıpları telafi etmek için her bir düğüm yerine bir rejenerasyon amplifikatörü gibi davranan akson boyunca bir düğümden diğerine etkili bir şekilde atlar. Akson yükseltisinde bir aksiyon potansiyeli yaratıldığında, aksondan diğer nöronlara iletilir.

Sinir sistemindeki sinyal iletiminin, bir nöronun tamamen aktif veya aktif olmadığı varsayılan dijital bir yapıya sahip olduğu sonucuna varmak mümkündür ama bu sonuç doğru değildir, çünkü bir nöron sinyalinin yoğunluğu impuls frekansında kodlanır. Daha iyi bir sonuç, biyolojik sinir sistemlerini, bilgi iletmek için bir nabız frekans modülasyonu kullanıyormuş gibi yorumlamak olacaktır. Belirli bir nöronun aksonu boyunca geçen sinir impulsları yaklaşık olarak sabit genliktedir, ancak üretilen impuls sayısı ve zaman aralıkları, nöronun yeterli uyarıcı girdilerin sinaptik kavşalara gelişiyle ilgili istatistiklerle kontrol edilir [13].

Biyofiziksel nöron çıkış davranışının temsili Şekil 4.5'te şematik olarak gösterilmiştir [14, 15]. T = 0 zamanında bir nöron uyarılır; T zamanında, tipik olarak 50 milisaniye olabilir, nöron aksonu boyunca bir impuls dizisi ateşlenir. Bu impulsların her biri pratik olarak aynı genliğe sahiptir. Bir süre sonra, $t = T + \tau$ civarında, nöron, aynı uyarmanın bir sonucu olarak başka bir impuls dizisini tetikleyebilir, ancak ikinci impuls dizisi genellikle daha küçük bir sayı içerecektir. Nöron uyarılmış olmasa bile, uyarılma durumdan çok daha az sıklıkta olsa da rastgele impuls gönderebilir.



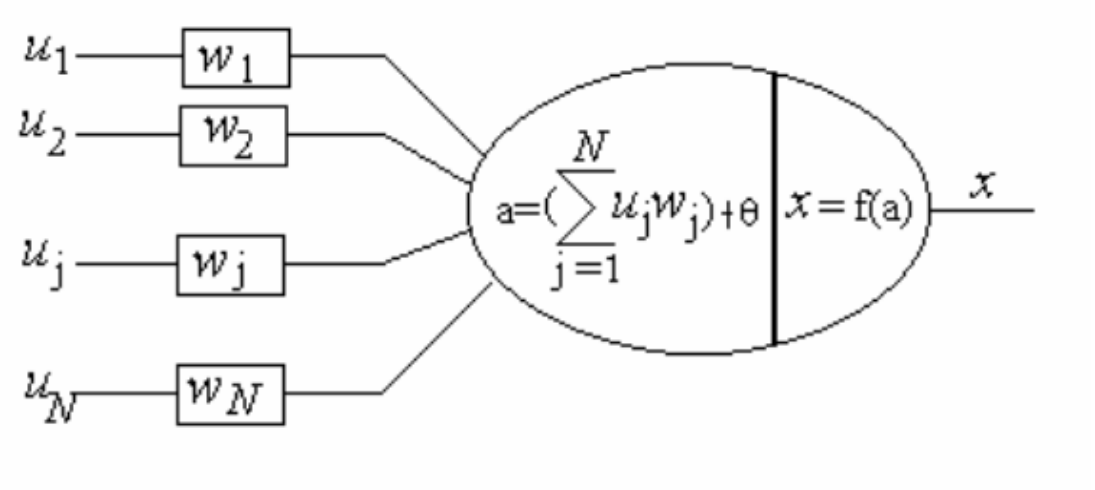
Şekil 4.5 : T = 0 zamanında uyarıldıktan sonra biyofiziksel nöron çıkış sinyalinin gösterimi.

Bir nöronun elektrokimyasal yapısını ve işleyişini açıklamak için önemli miktarda araştırma yapılmıştır, ancak gelecekte cevaplanması gereken birkaç soru olmaya devam etmektedir.

4.1 Yapay Nöron Modeli

Önceki bölümde bahsedildiği gibi sinapslar yoluyla sinyalin bir nörondan diğerine iletilmesi, belirli verici maddelerin kavşağın gönderici tarafından serbest bırakıldığı karmaşık bir kimyasal işlemdir. Etki, alıcı hücrenin gövdesi içindeki elektrik potansiyelini arttırmak veya azaltmaktır. Bu derecelendirilmiş potansiyel bir eşik

değere ulaşırsa, nöron ateşlenir. McCulloch ve Pitts tarafından önerilen yapay nöron modelinin [16] yeniden üretmeye çalışması bu özelliktir. Şekil 4.6'da gösterilen nöron modeli, yapay sinir ağlarında yaygın olarak kullanılan modeldir.



Şekil 4.6 : Yapay nöron.

Bu şekilde verilen yapay nöron, u_1, u_2, ... u_N olarak belirtilen N girişine sahiptir. Bu girişleri nörona bağlayan her satıra, sırasıyla w_1, w_2, ..., w_N olarak gösterilen bir ağırlık atanır. Yapay modeldeki ağırlıklar biyolojik nöronlardaki sinaptik bağlantılara karşılık gelir. Yapay nörondaki eşik genellikle θ ile temsil edilir ve dereceli potansiyele karşılık gelen aktivasyon formülle verilir:

$$a = \left(\sum_{j=1}^{N} w_j \, u_j \right) + \theta \tag{4.1}$$

Girişler ve ağırlıklar gerçek değerlerdir. Bir ağırlık için negatif bir değer engelleyici bir bağlantıyı, pozitif bir değer ise uyarıcı bir değeri gösterir. Biyolojik nöronlarda θ negatif bir değere sahip olmasına rağmen, yapay nöron modellerinde pozitif bir değer atanabilir. θ pozitifse, genellikle eşik olarak adlandırılır. Matematiksel kolaylığı için aktivasyon formülünde (+) işaretini kullanacağız. Bazen, eşik toplamı $u_0 = +1$ ve bağlantı ağırlığı $w_0 = \theta$ olduğu varsayılarak toplama kısmına basitlik için birleştirilir. Böylece aktivasyon formülü:

$$a = \sum_{j=0}^{N} w_j \, u_j \tag{4.2}$$

Nöronun çıkış değeri, biyolojik nöronların ateşleme frekansına benzer şekilde aktivasyonunun bir fonksiyonudur:

$$x = f(a) \tag{4.3}$$

Vektör gösterimi aşağıdaki gibidir.

$$a = \boldsymbol{w}^T \boldsymbol{u} + \theta \tag{4.4}$$

Vektör gösterimi bir nöronun aktivasyonunu ifade etmek için yararlıdır. Burada, u giriş vektörünün j'inci elemanı u_j'dir ve w ağırlık vektörünün j'inci elemanı w_j'dir. Bu vektörlerin her ikisi de N boyutundadır. w^Tu'nun w ve u vektörlerinin iç çarpımı olduğuna dikkat etmek gerekir, bu da skaler bir değer ile sonuçlanır. İç çarpım, eşit boyutlu vektörler üzerinde tanımlanan bir işlemdir. Bu vektörlerin birim uzunluğunda olması durumunda, iç çarpım bu vektörlerin benzerliğinin bir ölçüsüdür.

4.2 Aktivasyon Fonksiyonu

Aktivasyon fonksiyonu, bir YSA'nın gerçekten karmaşık bir şeyi öğrenmesi ve anlamlandırması için önemlidir. Ana amaçları, bir YSA'daki bir düğümün bir giriş sinyalini bir çıkış sinyaline dönüştürmektir. Bu çıkış sinyali, yığındaki bir sonraki katmana giriş olarak kullanılır.

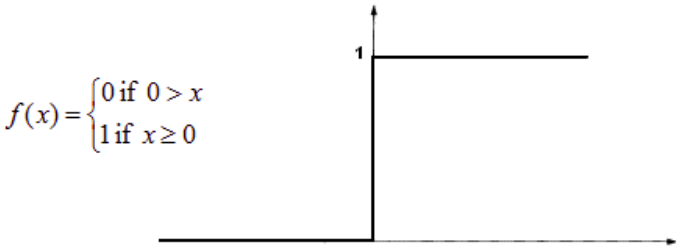
Aktivasyon fonksiyonu, ağırlıklı toplamı hesaplar ve bir eşik ekleyerek bir nöronun aktive edilip edilmeyeceğine karar verir. Amaç, bir nöronun çıkışına doğrusal olmayan bir hale getirmektir. Doğrusal olmayan fonksiyonlar, derecesi birden fazla olan ve eğriliğe sahip olan fonksiyonlardır.

Aktivasyon fonksiyonunu uygulamazsak, çıkış sinyali basitçe doğrusal fonksiyon olacaktır (birinci derece polinom). Doğrusal bir fonksiyonu çözmek kolaydır, ancak karmaşıklıkları sınırlıdır, daha az güce sahiptir. Aktivasyon fonksiyonu olmadan, yapay sinir ağı modelleri görüntüler, videolar, ses, konuşma vb. gibi karmaşık verileri öğrenemez ve modelleyemez.

Sinir ağları "Evrensel Fonksiyon Yaklaştırıcılar" olarak kabul edilir. Bu, herhangi bir işlevi hiç öğrenip hesaplayabilecekleri anlamına gelir.

4.2.1 Basamak aktivasyon fonksiyonu

Basamak fonksiyonu, eşik tabanlı bir etkinleştirme işlevidir (Şekil 4.7). Giriş değeri belirli bir eşiğin üstünde veya altındaysa, nöron etkinleştirilir ve bir sonraki katmana tam olarak aynı sinyali gönderir.



Şekil 4.7 : Basamak aktivasyon fonksiyonu.

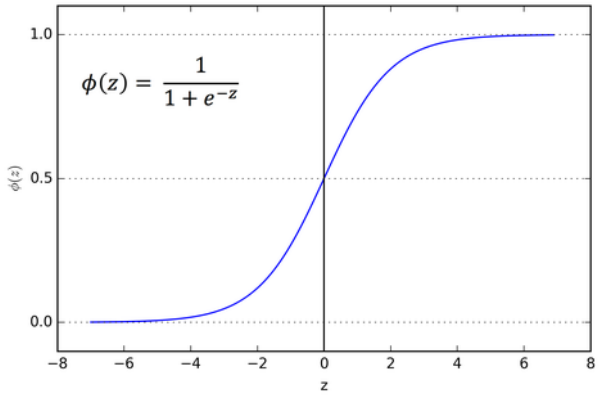
4.2.2 Sigmoid aktivasyon fonksiyonu

Sigmoid işlevi, 0 ile 1 arasında değişen karakteristik bir "S" şekilli eğri veya sigmoid eğriye sahip bir matematiksel işlevdir (Şekil 4.8), bu nedenle olasılığı bir çıktı olarak tahmin edilmesi gereken modeller için kullanılır.

Sigmoid işlevi farklıdır, yani eğrinin eğimini herhangi bir 2 noktada bulabileceğimiz anlamına gelir.

Sigmoid aktivasyon fonksiyonunun dezavantajı, güçlü negatif giriş sağlanırsa sinir ağının eğitim zamanında sıkışmasına neden olabilmesidir.

Sigmoid fonksiyon, yapay sinir ağlarının yapımında kullanılan en yaygın aktivasyon fonksiyonudur. Doğrusal ve doğrusal olmayan davranışlar arasında zarif bir denge sergileyen kesin olarak artan bir işlev olarak tanımlanır.

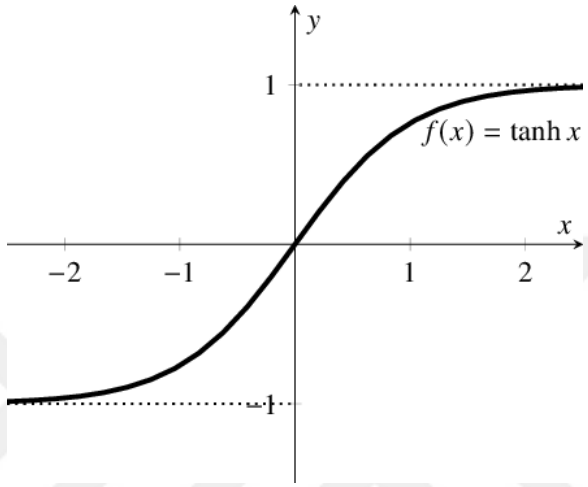


Şekil 4.8 : Sigmoid aktivasyon fonksiyonu.

4.2.3 Hiperbolik tanjant aktivasyon fonksiyonu

Sigmoid'e benzer, ancak performansta daha iyidir. Doğrusal değildir, bu nedenle katmanlar üst üste konabilir. Fonksiyon çıkışı (-1,1) arasında değişir (Şekil 4.9).

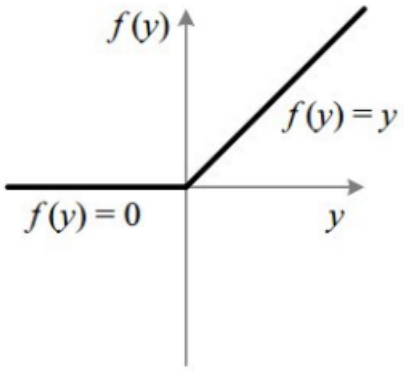
Bu fonksiyonun ana avantajı, güçlü negatif girişlerin negatif bir çıkışa eşlenmesidir ve sadece sıfır değerli girişlerin sıfıra yakın çıkışlara eşlenmesidir. Eğitim sırasında takılma olasılığı daha düşüktür.



Şekil 4.9 : Hiperbolik tanjant aktivasyon fonksiyonu.

4.2.4 ReLu (Rectified linear unit) aktivasyon fonksiyonu

ReLu, sıfırdan sonsuza kadar değişen ve en çok kullanılan aktivasyon fonksiyonudur (Şekil 4.10). Fonksiyonun girişi pozitifse, o değeri doğrudan çıkarır, aksi takdirde 0 çıktısı verir. ReLu doğrusal değildir ve ReLu'nun bir kombinasyonu da doğrusal değildir. Aslında, iyi bir tahmin edicidir ve herhangi bir fonksiyon ReLu'nun bir kombinasyonu ile yaklaştırılabilir.

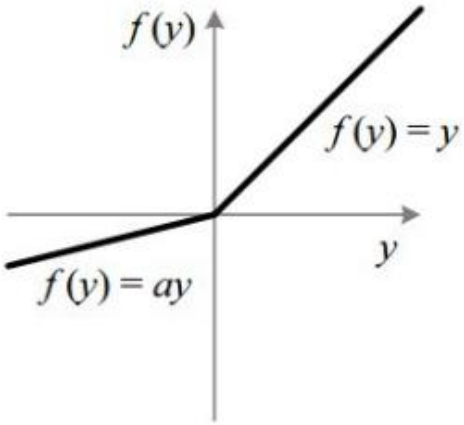


Şekil 4.10 : ReLu.

Sadece sinir ağının gizli katmanlarına uygulanmalıdır. Bu nedenle, çıkış katmanında da sınıflandırma problemi için softmax fonksiyonunu ve regresyon problemi için doğrusal fonksiyon kullanılmalıdır.

ReLu kullanıldığında karşılaşılabilecek problemlerden biri de şudur ki; yapay sinir ağlarının eğitimi sırasında bazı nöronlar ölebilir. Herhangi bir veri noktasında bir daha asla aktive olmayacak bir ağırlık güncellemesine neden olabilir. Kısaca ReLu ölü nöronlarla sonuçlanabilir.

Ölen nöronlar sorununu çözmek için Leaky ReLu tanıtılmıştır. Leaky ReLu güncellemeleri canlı tutmak için küçük bir eğim sunmaktadır ve a değeri 0.01 alınır (Şekil 4.11).



Şekil 4.11 : Leaky ReLu.

4.2.5 Softmax aktivasyon fonksiyonu

Matematikte, softargmax veya normalleştirilmiş üstel fonksiyon olarak da bilinen softmax fonksiyonu, K gerçek sayılarının bir vektörünü girdi olarak alan ve girdi sayılarının üstelleriyle orantılı K olasılıklarından oluşan bir olasılık dağılımına normalleştiren bir fonksiyondur (Şekil 4.12).



Şekil 4.12 : Softmax örneği.

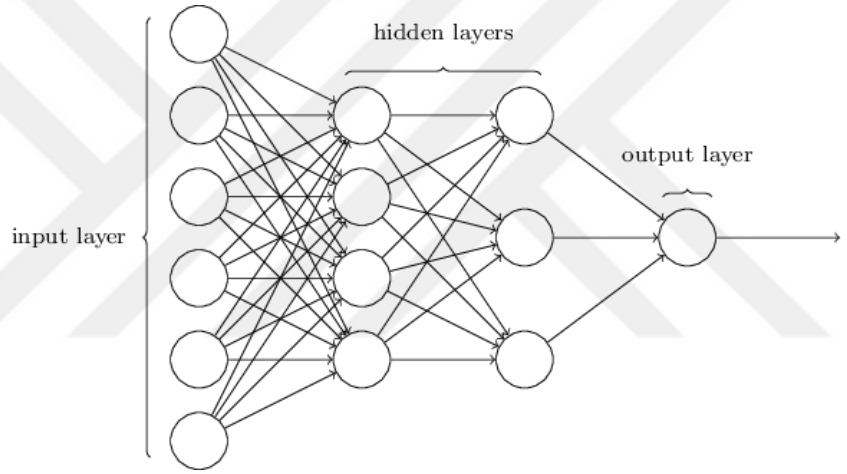
Formülü aşağıdaki gibidir.

$$f(x_j) = \frac{e^{x_j}}{\sum_{k=1}^{K} e^{x_k}} \tag{4.5}$$

Softmax fonksiyonu, sınıflandırma problemlerinde çokça tercih edilmektedir ve bu uygulamada da yapay sinir ağının çıkış katmanında kullanılacaktır.

4.3 Yapay Sinir Ağı Katmanları

Yapay sinir ağları tipik olarak giriş katmanları, gizli katmanlar ve çıkış katmanlarından oluşur (Şekil 4.13). Diğer bileşenler arasında kodlayıcı veya kod çözücü katmanlar bulunabilir.



Şekil 4.13 : İleri yönlü hesaplamalı ve geri yayılımlı bir ağ mimarisi.

4.3.1 Giriş katmanı

Bir sinir ağının giriş katmanı yapay giriş nöronlarından oluşur ve giriş verilerini yapay nöronların sonraki katmanları tarafından işlenmesi için sisteme getirir. Giriş katmanı, yapay sinir ağı için iş akışının başıdır.

Giriş katmanının belirgin özelliklerinden biri, giriş katmanındaki yapay nöronların farklı bir rol oynar çünkü giriş katmanı, önceki katmanlardan bilgi almayan "pasif" nöronlardan oluşmaktadır. Genel olarak, yapay nöronların bir dizi ağırlıklı girişe sahip olmaları ve bu ağırlıklı girdiler temelinde işlev görmeleri olasıdır; ancak teorik olarak bir giriş katmanı, ağırlıklı girişleri olmayan nöronlardan da oluşabilir.

Giriş katmanındaki nöronların sayısı, sisteme öğretilecek özniteliklerin sayısı ile aynı olmalıdır.

4.3.2 Gizli katman

Yapay sinir ağındaki gizli katman, giriş katmanları ve çıkış katmanları arasındaki bir katmandır; burada yapay nöronlar, bir dizi ağırlıklı girdi alır ve bir aktivasyon fonksiyonu yoluyla bir çıktı üretir. Mühendislerin insan beyninde devam eden aktivite türlerini simüle ettiği neredeyse tüm sinir ağlarının tipik bir parçasıdır.

Gizli sinir ağı katmanları birçok farklı şekilde kurulur. Bazı durumlarda, ağırlıklı girdiler rastgele atanır. Diğer durumlarda, geri yayılım adı verilen bir işlemle ince ayarlar ve kalibre edilirler. Her iki durumda da gizli katmandaki yapay nöron, beyindeki biyolojik bir nöron gibi çalışır. Giriş sinyallerini alır, üzerinde çalışır ve biyolojik nöronun aksonuna karşılık gelen bir çıktıya dönüştürür.

Makine öğrenimi modellerinin birçok analizi, sinir ağında gizli katmanların inşasına odaklanır. Çeşitli sonuçlar elde etmek için bu gizli katmanları ayarlamanın farklı yolları vardır. Örneğin, görüntü işlemeye odaklanan evrişimsel sinir ağları, bir bellek öğesi içeren tekrarlayan sinir ağları ve eğitim veri setlerinde doğrudan çalışan basit ileri beslemeli sinir ağları kullanılabilir.

4.3.3 Çıkış katmanı

Yapay bir sinir ağındaki çıkış katmanı, program için verilen çıktıları üreten son nöron katmanıdır. Sinir ağındaki diğer yapay nöronlara çok benzemelerine rağmen, çıkış katmanı nöronları, ağdaki son düğümler oldukları göz önüne alındığında, farklı bir şekilde inşa edilebilir veya gözlemlenebilir.

Tipik bir geleneksel sinir ağı üç tip katmana sahiptir: bir veya daha fazla giriş katmanı, bir veya daha fazla gizli katman ve bir veya daha fazla çıkış katmanı. Üç ayrı katmana sahip basit ileri beslemeli sinir ağları, temel anlaşılması kolay modeller sağlar. Daha sofistike, yenilikçi sinir ağları, her türden birden fazla katmana sahip olabilir ve belirtildiği gibi, her katman türü farklı şekilde inşa edilebilir. Geleneksel bir yapay nöron, bazı ağırlıklı girdilerden, bir dönüşüm fonksiyonundan ve biyolojik nöronun aksonuna karşılık gelen aktivasyon fonksiyonundan oluşur. Bununla birlikte, çıkış katmanı nöronları, yinelemeli sürecin nihai sonuçlarını düzene sokmak ve geliştirmek için farklı şekilde tasarlanabilir.

Bir anlamda çıktı katmanı birleşir ve somut olarak nihai sonucu üretir. Bununla birlikte, sinir ağını daha iyi anlamak için, giriş katmanına, gizli katmanlara ve çıkış katmanına bir bütün olarak bakmak önemlidir.

4.4 Yapay Sinir Ağlarında Öğrenme Stratejileri

Yapay sinir ağlarında öğrenme prosesi, nöronların ağırlıkları ya da sapma gibi serbest parametrelerin çevresel etkiler tarafından uyarılması ile değişmesi, kendini adapte etmesidir. Bu parametrelerin nasıl değişime uğradığı, öğrenme tipini tarafından belirlenir.

4.4.1 Danışmanlı (Supervised) öğrenme

Danışmanlı öğrenme, beklenen ve doğru bir şekilde etiketlenmiş çıkışları içeren eğitim veri setlerinin kullanıldığı bir öğrenme tipidir.

Danışmanlı öğrenim problemlerinde her şey doğru etiketlere sahip eğitim veri setleri ile başlar. Bu öğrenme stratejisi, el yazısıyla yazılan alfabe ve rakamların tanınmasına kullanılacak bir yapay sinir ağında uygulanabilir.

Danışmanlı öğrenmede, el yazısıyla yazılan alfabe veya rakamlardan oluşan binlerce resim verilerini, her bir görüntünün temsil ettiği etiketlerle alan danışmanlı bir öğrenme probleminde, denetimli öğrenme algoritmaları görüntüler arasındaki ilişkileri öğrenir ve daha sonra bu öğrenme ilişkisini sayılar veya alfabelerin yeni görüntülerini tanımlamak için uygular.

Danışmanlı öğrenme problemleri ayrıca regresyon ve sınıflandırma problemleri olarak gruplandırılabilir.

Bir sınıflandırma problemi, çıktı değişkeninin “kırmızı” veya “mavi” veya “hastalık var” ve “hastalık yok” gibi bir kategori olması durumudur.

Regresyon problemi, çıktı değişkeninin “dolar” veya “ağırlık” gibi gerçek bir değer olduğu durumudur.

4.4.2 Danışmansız (Unsupervised) öğrenme

Danışmansız öğrenmenin amacı, veriler hakkında daha fazla bilgi edinmek için verilerin temelini oluşturan yapıyı veya dağılımı modellemektir.

Danışmanlı öğrenmenin aksine eğitim için kullanılan veri setinde doğru cevaplar ve öğretmen yoktur. Algoritmalar, verilerdeki ilginç yapıyı keşfetmek ve temsil etmek için kendi kendilerine bırakılmıştır.

Denetimsiz öğrenme problemleri ayrıca kümeleme ve ilişkilendirme problemleri olarak gruplandırılabilir.

Kümeleme sorunu, satın alma davranışı yoluyla müşterileri gruplama gibi verilerdeki doğal gruplamaları keşfetmek istediği yerdir.

İlişkilendirme kuralı öğrenme sorunu, X satın alan kişiler de Y satın alma eğiliminde olduğu gibi verilerinizin büyük bölümlerini tanımlayan kuralları keşfetmek istediği yerdir.

4.4.3 Destekleyici (Reinforcement) öğrenme

Destekleyici öğrenme, insanların ve hayvanların çevre hakkında nasıl öğrendiklerini öğrenmelerine benzer. Takviye öğrenmede yapay sinir ağı, gerçekleştirdiği eylemler ve sonuçlar yoluyla öğrenir.

Destekleyici öğrenmede, öğrenci bir ortamda eylemde bulunan ve bir sorunu çözmeye çalışma eylemleri için ödül veya ceza alan bir karar verme aracısıdır. Deneme ve çalıştırma hata çalıştırmasından sonra, toplam ödülü en üst düzeye çıkaran eylem sırası olan en iyi politikayı öğrenmelidir.

Örneğin, satranç oynamayı bilen bir makine yapmak isteniyor. Bu senaryoda, danışmanlı öğrenme iki nedenden dolayı kullanılmamalıdır. İlk olarak, bir modelin oyun sayısı üzerinden eğitilmesi ve her pozisyon için her hareketi tanımlaması veya etiketlemesi için eğitmen bulundurmak çok maliyetlidir. İkincisi, birçok durumda hareket yoktur, önceki hamlelere bağlıdır. Tek geri bildirim, oyuncu oyunu kazandığında veya kaybettiğinde oyunun sonundadır.

Başka bir örnek, bir labirente yerleştirilen robottur. Robot yönlerden herhangi birinde hareket edebilir (ileri, geri, sağ veya sol) ve çıkışa ulaşmak için bir dizi hareket yapmalıdır. Robot labirentte olduğu sürece, geri bildirim yoktur ve robot çıkışa ulaşmak için birçok hareket dener. Bu durumda rakip yoktur, ancak bu durumda zamana karşı oynandığını ima eden daha kısa yörüngeler tercih edebilir.

Destekleyici öğrenmede sahip olunan bir eğitmenle öğrenmenin aksine “eleştirmenle öğrenme” de denebilir.

4.5 Yapay Sinir Ağı Mimarileri

Nöronları tekil olarak kullanmak çoğu zaman bir problemi çözmek için yeterli değildir. Bu yüzden birden fazla nöron kullanılarak bir yapay sinir ağı mimarisi oluşturulur ve bu sayede karışık problemleri çözmek için kullanılabilecek bir yapay sinir ağı modeli elde edilmiş olunur.

4.5.1 İleri beslemeli yapay sinir ağı modeli

Bunlar pratik uygulamalarda en yaygın sinir ağı türüdür. İlk katman girdidir ve son katman çıktıdır. Birden fazla gizli katman varsa, bu modellere "derin" sinir ağları denir. Durumlar arasındaki benzerlikleri değiştiren bir dizi dönüşüm hesaplarlar. Her bir katmandaki nöronların aktiviteleri, bir önceki katmandaki aktivitelerin doğrusal olmayan bir fonksiyonudur.

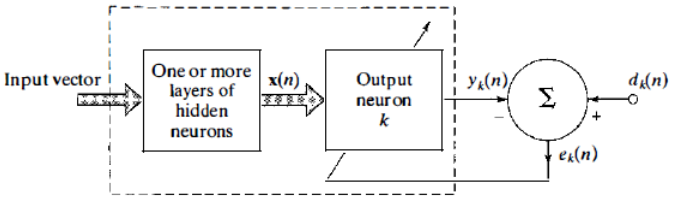
4.5.2 Geri beslemeli yapay sinir ağı modeli

Bunlar, bağlantı grafiğindeki yönlendirilmiş döngülere sahiptir. Bu, bazen okları izleyerek başladığınız yere geri dönebileceğiniz anlamına gelir. Karmaşık dinamiklere sahip olabilirler ve bu onların eğitilmesini çok zorlaştırabilir. Biyolojik olarak daha gerçekçi modellerdir.

Şu anda tekrarlayan ağları eğitmenin etkili yollarını bulmak için çok fazla ilgi vardır. Tekrarlayan sinir ağları, sıralı verileri modellemenin çok doğal bir yoludur. Zaman dilimi başına bir gizli katman ile çok derin ağlara eşdeğerdirler; ancak her zaman diliminde aynı ağırlıkları kullanırlar ve her zaman diliminde girdi alırlar. Gizli durumlarındaki bilgileri uzun süre hatırlama yeteneğine sahiptirler, ancak bu potansiyeli kullanmak için onları eğitmek çok zordur.

4.6 Hata Düzeltme Tabanlı Öğrenme

Şekil 4.13'de gösterilen blok diyagramı çok katmanlı bir YSA'ya aittir.



Şekil 4.14 : YSA blok diyagramı.

Çıkış nöronuna **x(n)** sinyali gelmektedir ve bu sinyal de bir veya birden fazla gizli katman tarafından üretilmektedir. Gizli katmanlara da giriş katmanından gelen veriler ile beslenmektedir. **n** ile belirtilen sayı ise **k** nöronuna ait ağırlıkların ayarlandığı iterasyon adımını temsil etmektedir. Çıkış nöronuna ait sinyal ile $y_k(n)$ ile gösterilmektedir. $d_k(n)$ ile belirtilen beklenen değer ile nörona ait çıkış sinyali birbirinden çıkartılarak hata sinayli elde edilir.

$$e_k(n) = d_k(n) - y_k(n) \tag{4.6}$$

Elde edilen hata sinyali $e_k(n)$ çok önemlidir çünkü sinaptik ağırlıklarda yapılacak olan düzeltmeler bu hata sinyalini azaltmaya yönelik yapılacaktır. Adım adım yapılacak olan bu düzeltmelerin amacı çıkış nöronundaki çıkış sinyali ile beklenen sinyal arasındaki farkı azaltmaktır. Bunun için de bir maliyet fonksiyonu kullanılmalıdır.

$$\mathcal{E}(n) = \frac{1}{2}\, e_k^2(n) \tag{4.7}$$

Denklem 4.7'de verilmiş olan maliyet fonksiyonuna hatanın anlık enerjisi de denebilir. Adım adım sinaptik ağırlıkların ayarlanması işlemi, sinaptik ağırlıklar kararlı duruma gelene kadar devam eder. Bu noktada öğrenme sonlanır.

Maliyet fonksiyonunun minimize edilmesi delta kuralı veya Widrow-Hoff kuralı olarak bilinmektedir. $x(n)$ giriş sinyalinin **n** zamanında **k** numaralı nöronu uyarırken hesaplamada kullanılan sinaptik ağırlık $\omega_{kj}(n)$ ile gösterilmektedir. Bu kurala göre **n** zamanında uygulanacak sinaptik ağırlık değişimi $\Delta\omega_{kj}(n)$ ile gösterilmektedir ve aşağıdaki gibi hesaplanabilir.

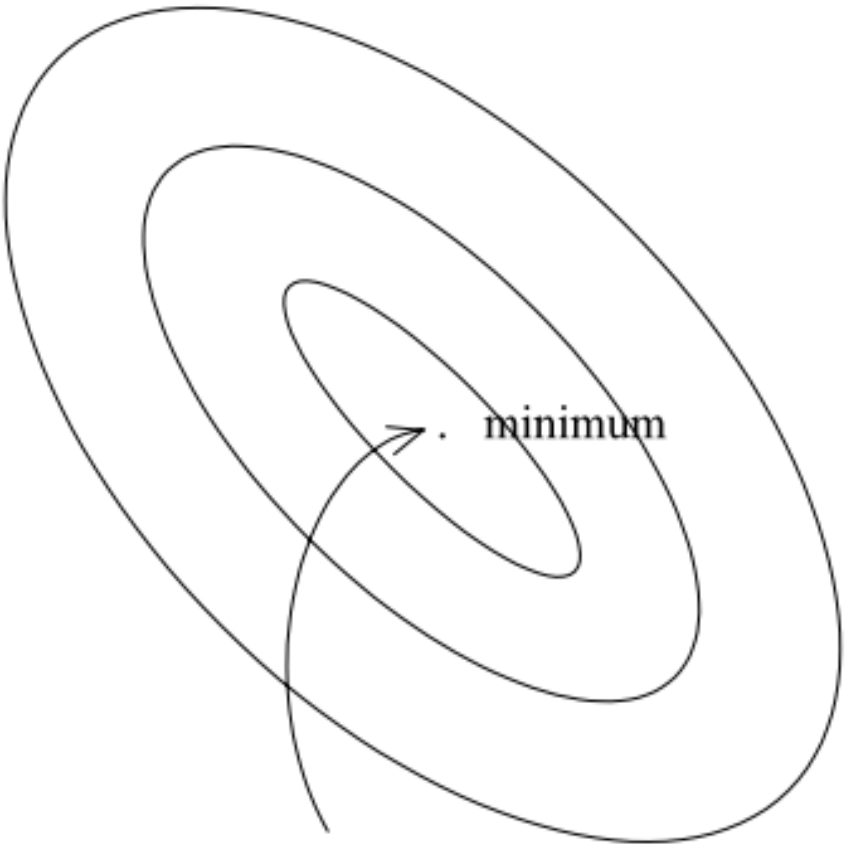
$$\Delta\omega_{kj}(n) = \eta e_k(n) x_j(n) \tag{4.8}$$

Denklem 4.8'de η sembolü öğrenme oranını temsil etmektedir. Bu kuralın uygulanabilmesi için hata sinyalinin ölçülebilir olması çok önemlidir. Sinaptik ağırlıkların güncellenmesi aşağıdaki gibi yapılmaktadır.

$$\omega_{kj}(n+1) = \omega_{kj}(n) + \Delta\omega_{kj}(n) \tag{4.9}$$

4.7 En Dik Gradyan Azalma Yöntemi

Amaç maliyet fonksiyonunu minimize etmektir. Bunun için maliyet fonksiyonunun gradyanı hesaplanır ve bu gradyana tam ters yönde sinaptik ağırlıkların güncellemesi yapılır (Şekil 4.15).

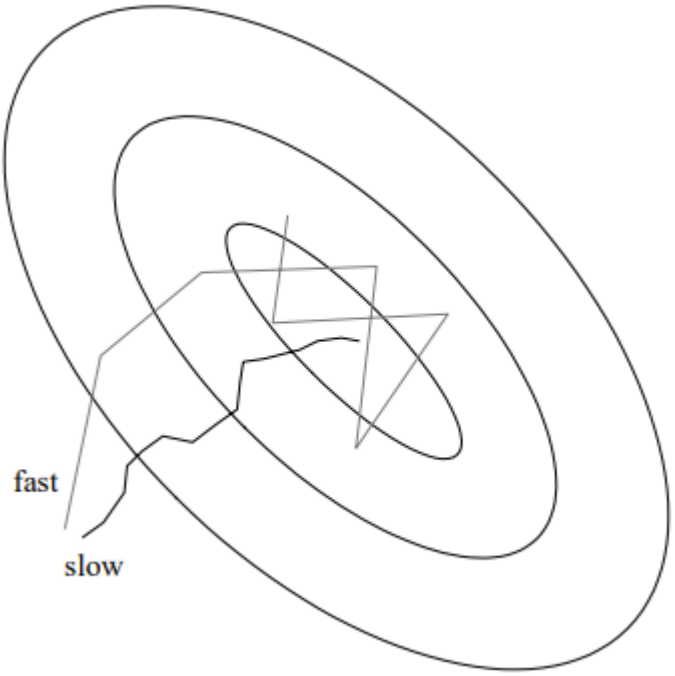


Şekil 4.15 : İki boyutlu nöron ağırlıkları uzayında hatanın gösterimi.

Öğrenme oranının seçimi çok önemlidir. Eğer çok küçük seçilirse hatayı minimize eden değerlerin bulunması çok uzun sürebilir (Şekil 4.16). Çok büyük seçilirse ise hiçbir zaman bulamayabilir. Ayrıca öğrenme oranının seçimine göre sinaptik ağırların lokal minimumda takılma ihtimali de vardır.

Bu yöntem zikzaklar çizdiği için yavaş bir yöntemdir. Eğer hatanın kontörleri bir hiperküre oluşturuyorsa, en dik azalma yöntemi global minimuma götürür. Aksi takdirde, global minimuma götürmesinin bir garantisi yoktur.

En dik azalma yöntemi, gradyan azalma yöntemlerinden biridir ve birçok gradyan azalma yöntemi bulunmaktadır. Bu uygulama **Adam** optimizasyon yöntemi kullanılacaktır.



Şekil 4.16 : Öğrenme oranının etkisi.

4.8 Geri Yayılım

Delta kuralını temel alan ve çok katmanlı yapay sinir ağı modellerinde sinaptik ağırlıkların güncellemesi için kullanılan yöntemdir. Amaç sinaptik ağırların değerlerini maliyet fonksiyonunu minimize edecek şekilde güncellemektir.

$$E(n) = \sum_{k} \varepsilon(n)^2 \tag{4.10}$$

$$\Delta\omega_{kj}(n) = -\eta \, \frac{\partial E(n)}{\partial \omega_{kj}(n)} \tag{4.11}$$

Denklem 4.11'de her bir ağırlığa göre zincir türevi alınmıştır.

5. ÖRNEK GÖRÜNTÜ İŞLEME UYGULAMASI

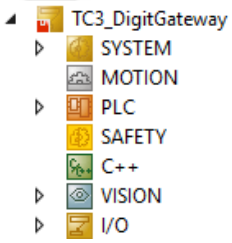
5.1 Uygulamanın Amacı

Endüstriyel kamera vasıtasıyla alınan görüntülerdeki veya çevrimdışı dosya kaynaklarından sağlanan görüntülerdeki el yazısı ile yazılmış rakamları yapay sinir ağı kullanarak kategorize etmektir. Endüstriyel kameradan alınan görüntüler, endüstriyel bilgisayarda ön işlemeye tabi tutulduktan sonra yapay sinir ağı modeline gönderilecektir ve sonuçlar arayüz yardımı ile görüntülenecektir.

5.2 TwinCAT Ortamının Hazırlanması

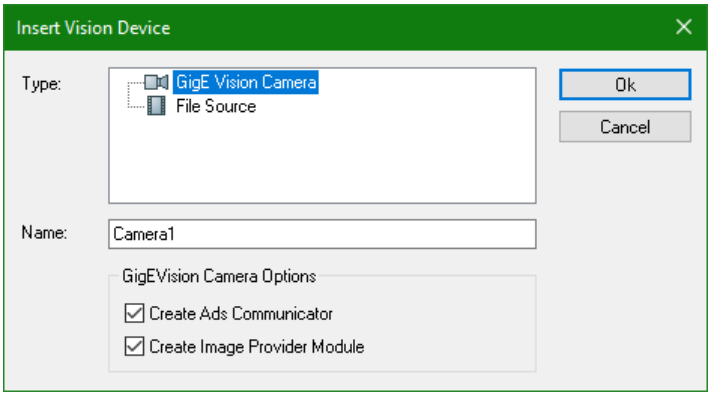
5.2.1 Endüstriyel kameranın taratılması ve konfigürasyonu

Şekildeki modüllerden "VISION" ile isimlendirilmiş modüle sağ tıklanır ve yeni bir obje eklenir (Şekil 5.1).



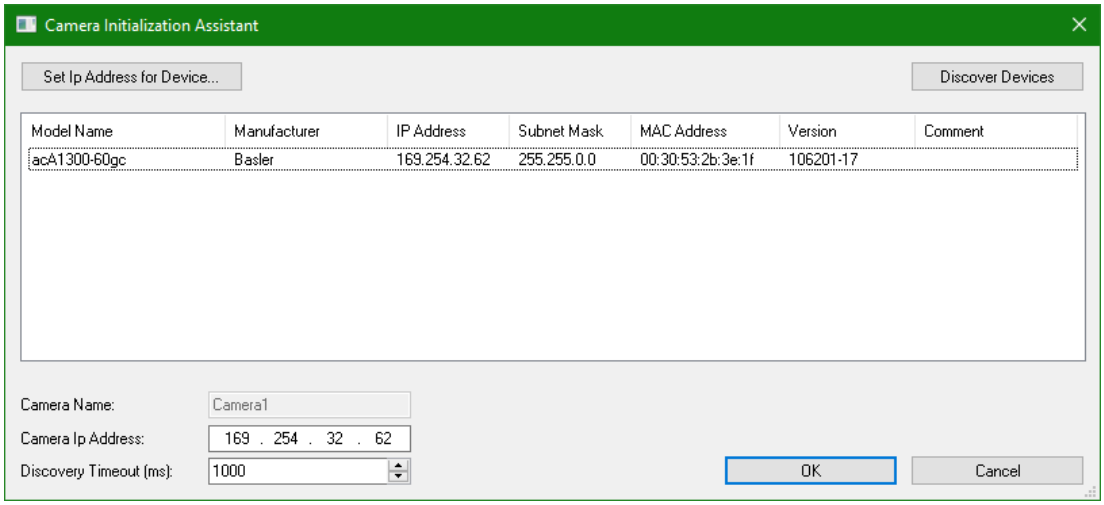
Şekil 5.1 : PLC projesinin modülleri.

GigE uyumlu kamera eklenir (Şekil 5.2).



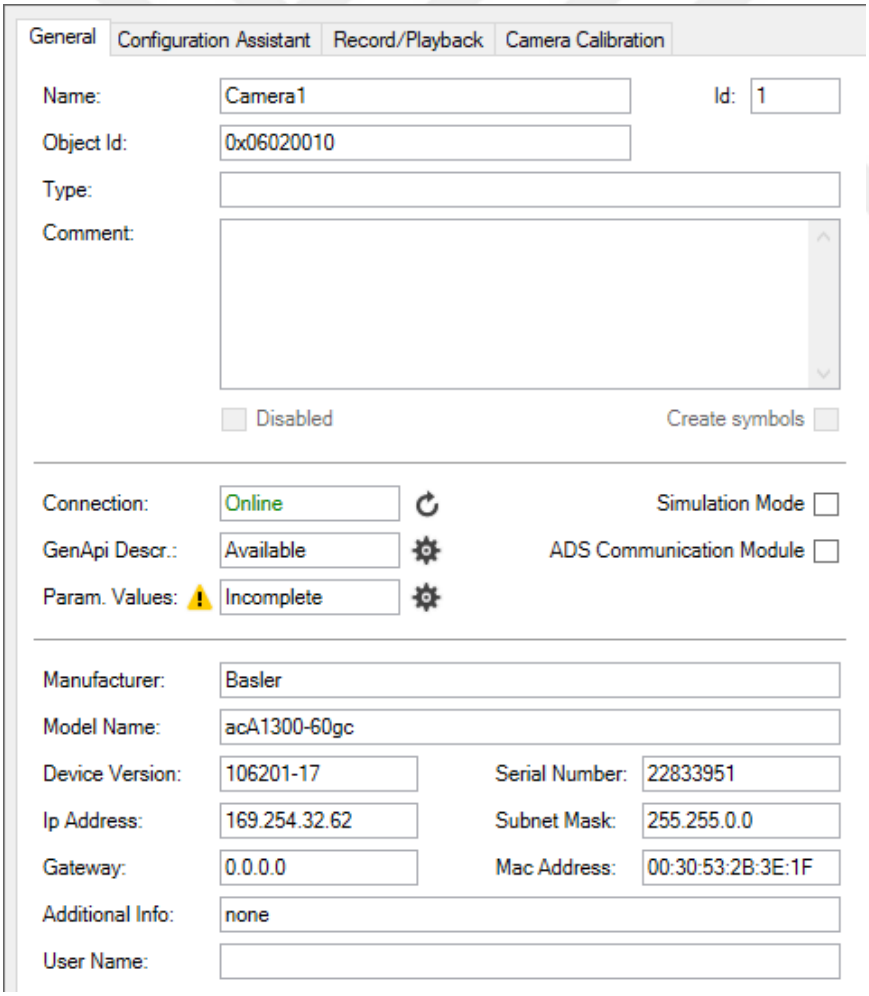
Şekil 5.2 : Kaynak seçimi.

GigE Vision standardıyla uyumlu kameralar otomatik olarak bulunabilir (Şekil 5.3).



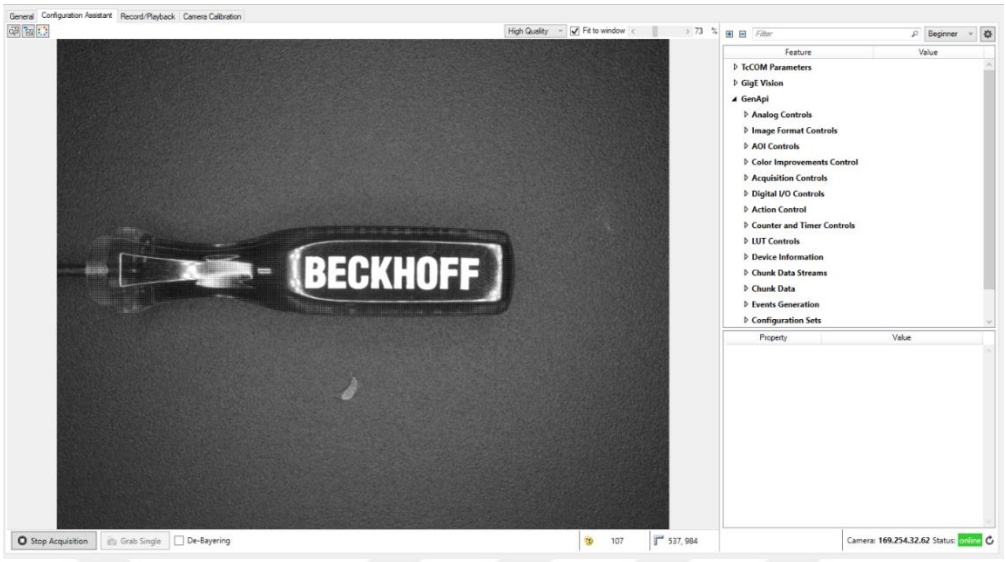
Şekil 5.3 : Endüstriyel kameranın taratılması.

Kamera eklendikten sonra gerekli konfigürasyonlar yapılabilir (Şekil 5.4).



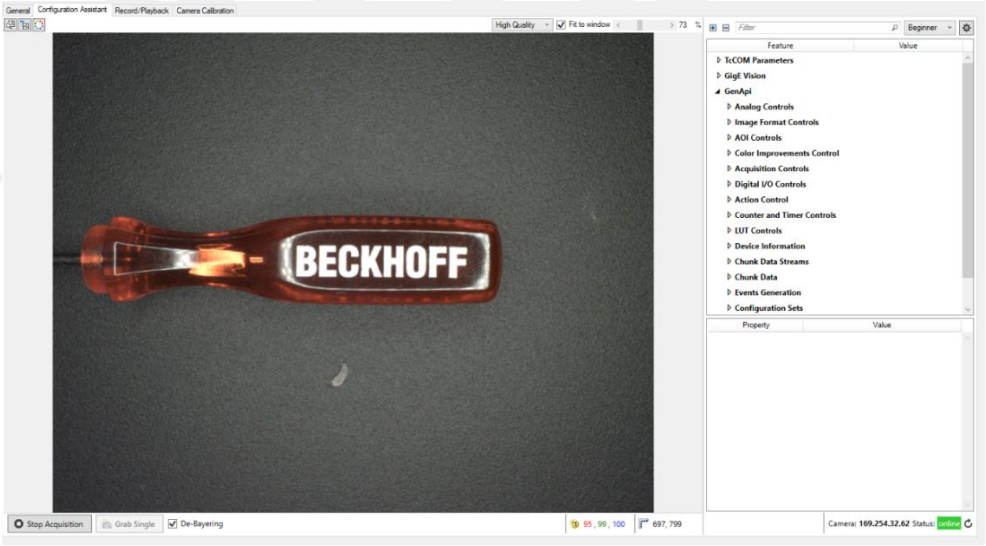
Şekil 5.4 : Endüstriyel kameranın konfigüre edilmesi.

“Configuration asistant” kullanılarak görüntü test edilebilir (Şekil 5.5).



Şekil 5.5 : Endüstriyel kameranın test edilmesi.

Renkli olarak test etmek de mümkündür (Şekil 5.6).

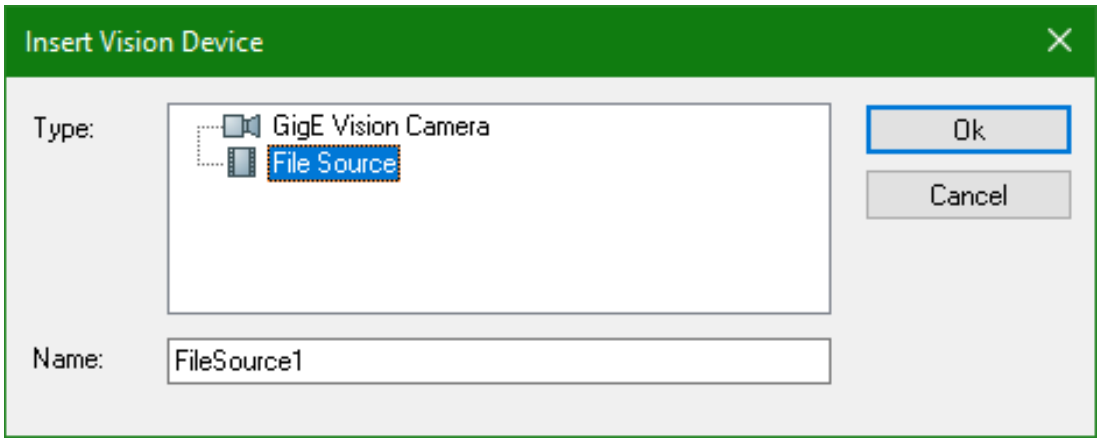


Şekil 5.6 : Endüstriyel kameranın renkli test edilmesi.

Endüstriyel kamera projeye eklendikten ve gerekli konfigürasyonlar yapıldıktan sonra PLC projesinde kullanılmaya hazırdır; ancak endüstriyel kameranın olmadığı durumlarda, çevrimdışı dosya kaynakları ile çalışmak mümkündür.

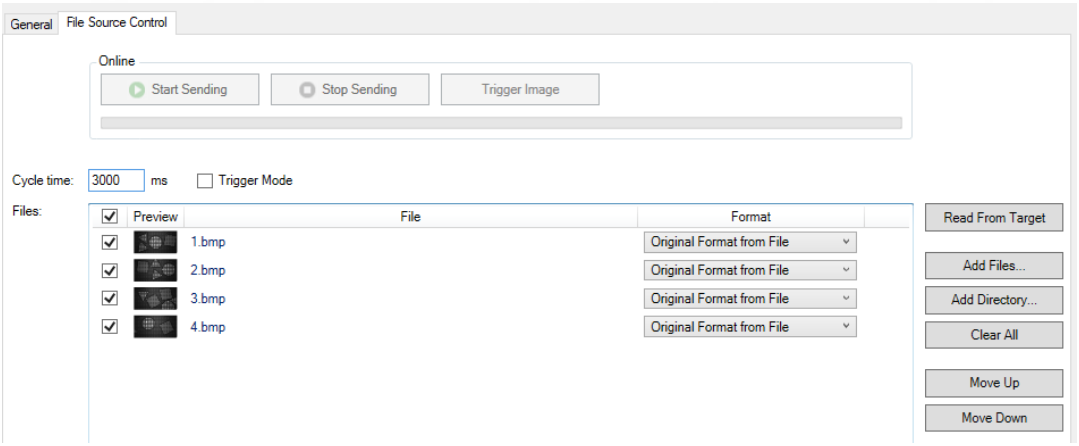
5.2.2 Çevrimdışı dosya kaynakları ile çalışma

Her zaman endüstriyel kamera ile çalışmak mümkün olmayabilir. Böyle durumlarda çevrimdışı dosya kaynakları kullanılabilir. Öncelikle projeye şekildeki gibi bir dosya kaynağı eklenmelidir (Şekil 5.7).



Şekil 5.7 : Dosya kaynağı cihazı ekleme.

Kullanılacak dosyaların eklenmesi (Şekil 5.8).



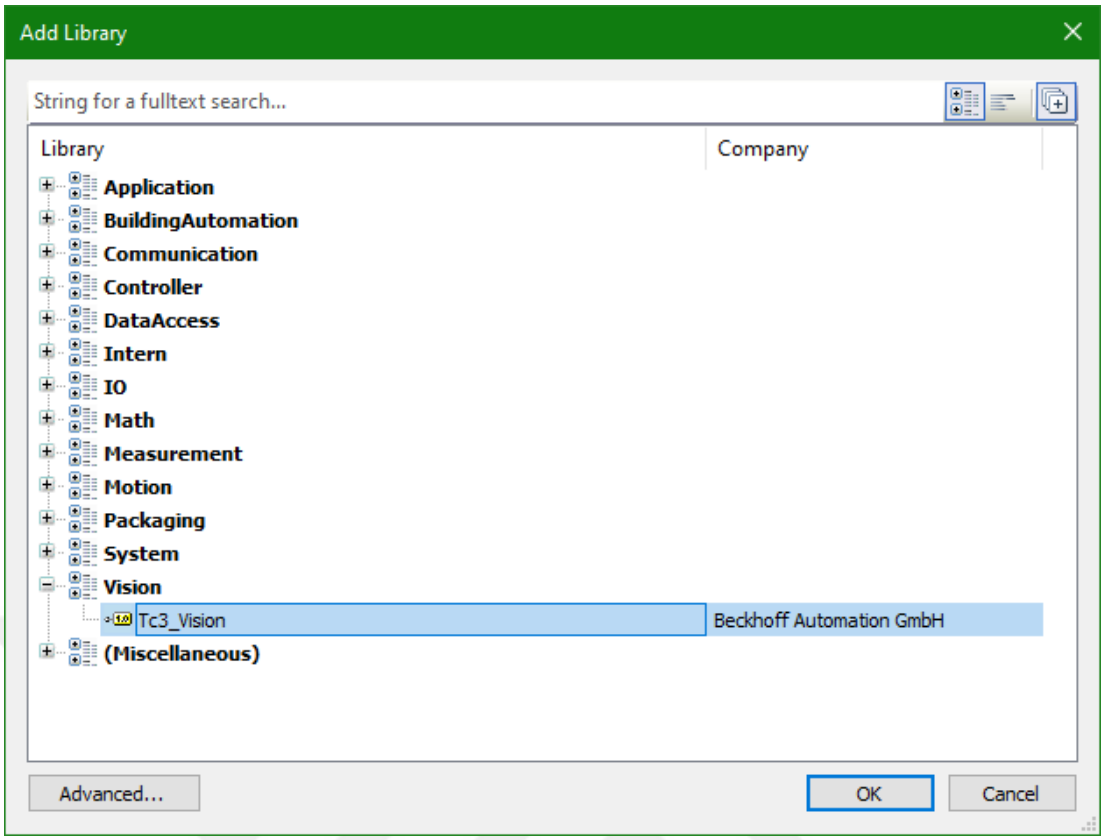
Şekil 5.8 : Dosyaların eklenmesi.

Dosya kaynağı projesi eklendikten sonra yukarıdaki şekildeki dosya ekleme butonuna basılarak istenilen fotoğraflar eklenebilir.

5.2.3 PLC projesinin hazırlanması

Bu bölümde ise daha önceden eklemiş olduğumuz kamera veya dosya kaynağı objelerinin PLC kodu içerisinde nasıl kullanılması gerektiğine değinilecektir.

Öncelikle şekil 5.1'deki modüllerden PLC modülüne proje eklenmelidir. Daha sonrasında projenin referanslarına "Tc3_Vision" kütüphanesi eklenmelidir (Şekil 5.9). Daha önceden de bahsedildiği gibi, bu kütüphane görüntü işleme algoritmalarının bulunduğu kütüphanedir ve kamera veya dosya kaynağı ile haberleşmenin kurulacağı objeleri içermektedir. Neredeyse tüm görüntü algoritmaları mevcuttur ve tamamen PLC ortamı içinde kod yazılmaktadır.



Şekil 5.9 : Vision kütüphanesinin eklenmesi.

Kütüphane eklendikten sonra PLC kodu yazılmaya hazırdır. Örneğin çevrimdışı dosya kaynağı ile çalışmak isteniyorsa, şekildeki gibi değişkenler tanımlanabilir (Şekil 5.10).

```
VAR
    fbCameraControl         : FB_VN_GevCameraControl;
    eState                  : ETcVnCameraState;

    hr                      : HRESULT;

    ipImageIn               : ITcVnImage;
    ipImageInDisp           : ITcVnDisplayableImage;

    ipImageOut              : ITcVnImage;
    ipImageOutDisp          : ITcVnDisplayableImage;

END_VAR
```

Şekil 5.10 : PLC değişkenleri

Eğer kamera kullanılmak isteniyor ise "FB_VN_FileSourceControl" yerine "FB_VN_GevCameraControl" fonksiyon bloğu kullanılmalıdır.

Her bir kameranın veya dosya kaynağının kendi durumları vardır ve bu durumları PLC içinde temsil etmek için "ETcVnCameraState" isimli bir yapı kullanılır (Şekil 5.11).

```
(* Specifies the state of a camera controller, which controls an attached camera. *)
    {attribute 'TcIecLibraryType'}
{attribute 'Namespace' := 'Tc3_Vision'}
TYPE ETcVnCameraState : (
    TCVN_CS_ERROR:=-1,  (* The camera controller is in an error state (e.g. the connection to the camera was lost). *)
    TCVN_CS_INITIAL:=0,     (* The camera controller is in the initial state and ready to establish the connection to the camera. *)
    TCVN_CS_INITIALIZING:=1,    (* The camera controller is initializing the camera. *)
    TCVN_CS_INITIALIZED:=2,     (* The camera controller has initialized the camera. *)
    TCVN_CS_OPENING:=3,     (* The camera controller is establishing the connection to the camera. *)
    TCVN_CS_OPENED:=4,  (* The connection to the camera has been established and the camera controller is ready to start the image acquisition. *)
    TCVN_CS_STARTACQUISITION:=5,    (* The camera controller is starting the image acquisition. *)
    TCVN_CS_ACQUIRING:=6,   (* The camera is sending images (either streaming or manually triggered, depending on the configuration). *)
    TCVN_CS_STOPACQUISITION:=7,     (* The camera controller is stopping the image acquisition. *)
    TCVN_CS_RESETTINGFEATURES:=8,   (* The camera controller is resetting the features on the camera. *)
    TCVN_CS_TRIGGERING:=9,  (* The camera controller is processing a softwaretrigger. *)
    TCVN_CS_CLOSING:=10 (* The camera controller is closing the connection to the camera. *)
) DINT;
END_TYPE
```

Şekil 5.11 : Kamera durumları.

PLC uygulamasında her çevrimde dosya kaynağının durumunu almak gereklidir. Alınan bu duruma göre bir durum makinesi oluşturulmalıdır (Şekil 5.12).

```
eState := fbCameraControl.GetState();

CASE eState OF
    TCVN_CS_ERROR:
        fbCameraControl.Reset();

    TCVN_CS_INITIAL,
    TCVN_CS_OPENING,
    TCVN_CS_OPENED,
    TCVN_CS_STARTACQUISITION:
        hr := fbCameraControl.StartAcquisition();

    TCVN_CS_ACQUIRING:
        // Normal çalışmada görüntü işleme kodu buraya eklenecek
        ;
    TCVN_CS_STOPACQUISITION:
        ;
    TCVN_CS_TRIGGERING:
        ;
    TCVN_CS_CLOSING:
        ;

END_CASE
```

Şekil 5.12 : Durum makinesi.

Yukarıdaki durum makinesi kullanılabilir. Burada “hr” olarak belirtilmiş olan değişken, fonksiyonların sıra ile çalışmasını sağlamaktadır. Örneğin kameradaki görüntüyü almak ve bunu ADS Image Watch objesinde izlemek için, aşağıdaki gibi bir kod yazılabilir (Şekil 5.13).

```
TCVN_CS_ACQUIRING:
    hr := fbCameraControl.GetCurrentImage(ipImageIn);
    hr := F_VN_CopyIntoDisplayableImage(ipImageIn, ipImageInDisp, hr);
```

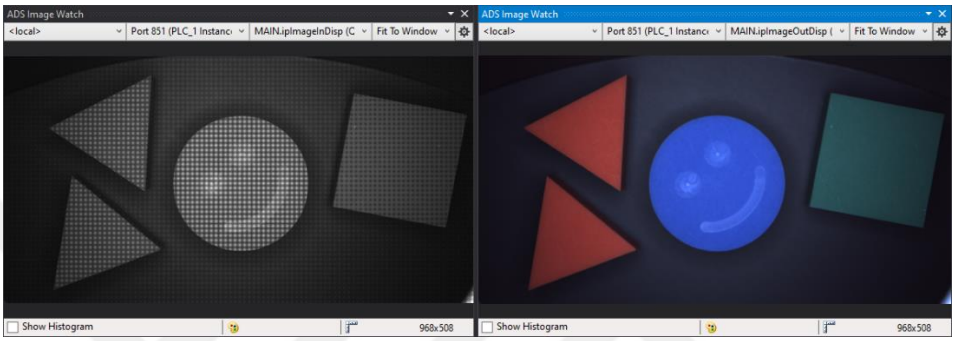
Şekil 5.13 : Örnek kod parçası.

Alınacak olan görüntü siyah beyazdır ve “F_VN_ConvertColorSpace” fonksiyonu kullanarak renkli yapmak mümkündür. “F_VN_CopyIntoDisplayableImage” fonksiyonu yardımı ile de bu görüntüler izlenebilir hale getirilir (Şekil 5.14).

```
hr := F_VN_ConvertColorSpace(ipImageIn, ipImageOut, ETcVnColorSpaceTransform.TCVN_CST_BAYER_RG_TO_RGB, hr);
hr := F_VN_TransformIntoDisplayableImage(ipImageOut, ipImageOutDisp, hr);
```

Şekil 5.14 : Örnek kod parçası.

İzlenebilir hale getirilen görüntüler, aşağıdaki gibi ADS Image Watch yardımı ile izlenebilir (Şekil 5.15).



Şekil 5.15 : Görüntülerin izlenmesi.

5.2.4 Görüntüye uygulanan ön işlem

Bu uygulamada kamera ile görüntü alınabileceği gibi, dosya kaynakları da kullanılabilir; ancak görüntünün kaynağı fark etmeksizin, PLC tarafında bazı görüntü işleme adımları izlenecektir. Bu adımlar, görüntüleri oluşturan piksellerin yapay sinir ağına göndermeden önce PLC içerisinde yapılmaktadır ve sırasıyla aşağıdaki gibidir.

- Renkli tonlardan gri tona çevirme
- Eşik uygulayarak grilerin beyaza dönüştürülmesi
- Görüntü üzerindeki en büyük objenin algılanması
- En büyük objenin ilgi bölgesi seçilmesi
- Görüntünün 28x28 piksele ölçeklendirilmesi

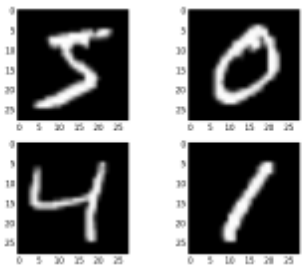
PLC kodu çalıştırıldığında görüntü kaynağından görüntünün gelip gelmediğini sürekli sorgulamaktadır. Eğer görüntü gelirse, ilk işlem olarak “F_VN_ConvertColorSpace” fonksiyonu ile renkliden gri tonlara çevirme yapılır. Daha sonra “F_VN_Threshold” uygulanarak gri tonlar beyaz yapılır. Bunu yapmaktaki amaç şudur; yapay sinir ağını eğitmek için kullandığımız veriler beyaz veya siyaha çok yakın tonlardan oluşmaktadır

ve gri tonlar gürültü olarak algılanmaktadır. Yapay sinir ağının sağlıklı bir şekilde resmi sınıflandırması için, bu gürültüden kurtulmak gereklidir. Sonrasında "F_VN_FindContoursExp" fonksiyonu kullanılarak resim üzerindeki objelerin hepsi taranır ve hafızaya alınır. Bunu yapmaktaki amaç ise alınan resimdeki yazı normalden ufak veya normalden büyük olabilir ve böyle bir durumda resmi normalleştirmek gerekmektedir. Tüm elemanlar bulunduktan sonra "F_VN_GetForwardIterator" fonksiyonu yardımıyla bulunan elemanlar teker teker taranır ve "F_VN_ContourArea" fonksiyonu kullanılarak aralarından en büyük alana sahip olan belirlenir. En büyük objeyi kapsayacak bir dikdörtgen "F_VN_UprightBoundingRectangle" fonksiyonu ile çizilir; ancak çizilmeden önce hem genişlik olarak hem de yükseklik olarak boşluk bırakılır. Bunu yapmaktaki amaç, yine elde edilen resmi eğitim verilerine benzetmektir. "F_VN_SetRoi_TcVnRectangle_UDINT" fonksiyonu ile ilgi alanı (region of interest) belirlenir. Sonuç olarak elde edilen resim yüksek çözünürlüklü olabilir. Yapay sinir ağına 28x28 piksel boyutlarında bir görüntü vermek zorunludur; bu yüzden de "F_VN_ResizeImage" fonksiyonu ile görüntü ölçeklendirilir. Son olarak da "F_VN_ExportImage" fonksiyonu ile bu görüntü PLC değişkenine aktarılır.

5.3 Python İle Yapay Sinir Ağları

Yapay sinir ağının matematiksel modelinin oluşturulması ve eğitimi için Python yazılım dili kullanılacaktır. Bu programlama dili özellikle veri işleme ve yapay sinir ağları gibi matematiksel olarak ağır uygulamalarda kullanılmaya çok uygundur. Yapay sinir ağı sıfırdan yazılabilir; ancak TensorFlow gibi kütüphaneler sayesinde yapay sinir ağının modellemesi ve eğitimi kolaylıkla yapılabilir.

Kütüphane içinde kullanılmaya hazır ve el yazısıyla yazılmış rakamlardan oluşan fotoğraflar mevcuttur. Bu projede 60000 tane fotoğraf YSA'yı eğitmek için, 10000 tane fotoğraf ise YSA'yı test etmek için kullanılacaktır (Şekil 5.16).

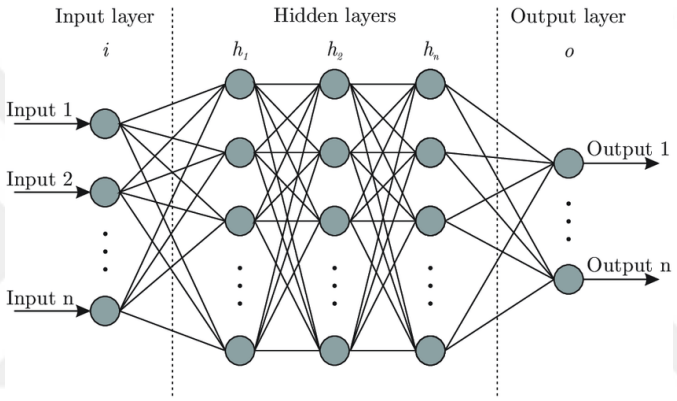


Şekil 5.16 : Elyazısıyla yazılmış rakamlar.

Amaç evrişimsel yapay sinir ağı kullanmak; ancak yığın boyutu (batch size), döngü sayısı (epoch), kaç tane gizli katman olacağı, bu katmanlarda kaç tane nöron olacağı gibi değerlerin belirlenmesi lazım.

Ek 1'de verilmiş olan kod kullanılarak bazı grafikler elde edilmiştir ve bir sonraki bölümlerde, bu grafikler üzerinde durulacaktır. İşlem kolaylığı sağlaması açısından basit bir yapay sinir ağı yapısı kullanılmıştır.

Aksi belirtilmediği takdirde, yapılacak olan denemelerde sadece bir tane değişen parametre olacaktır ve diğer parametreler sabit tutulacaktır. Varsayılan değerler şu şekildedir; veri setinin boyutu 10000, döngü sayısı 50, yığın sayısı 32, gizli katman sayısı 2 ve bu katmanlarda 64 adet nöron vardır (Şekil 5.17).



Şekil 5.17 : Temsili YSA modeli.

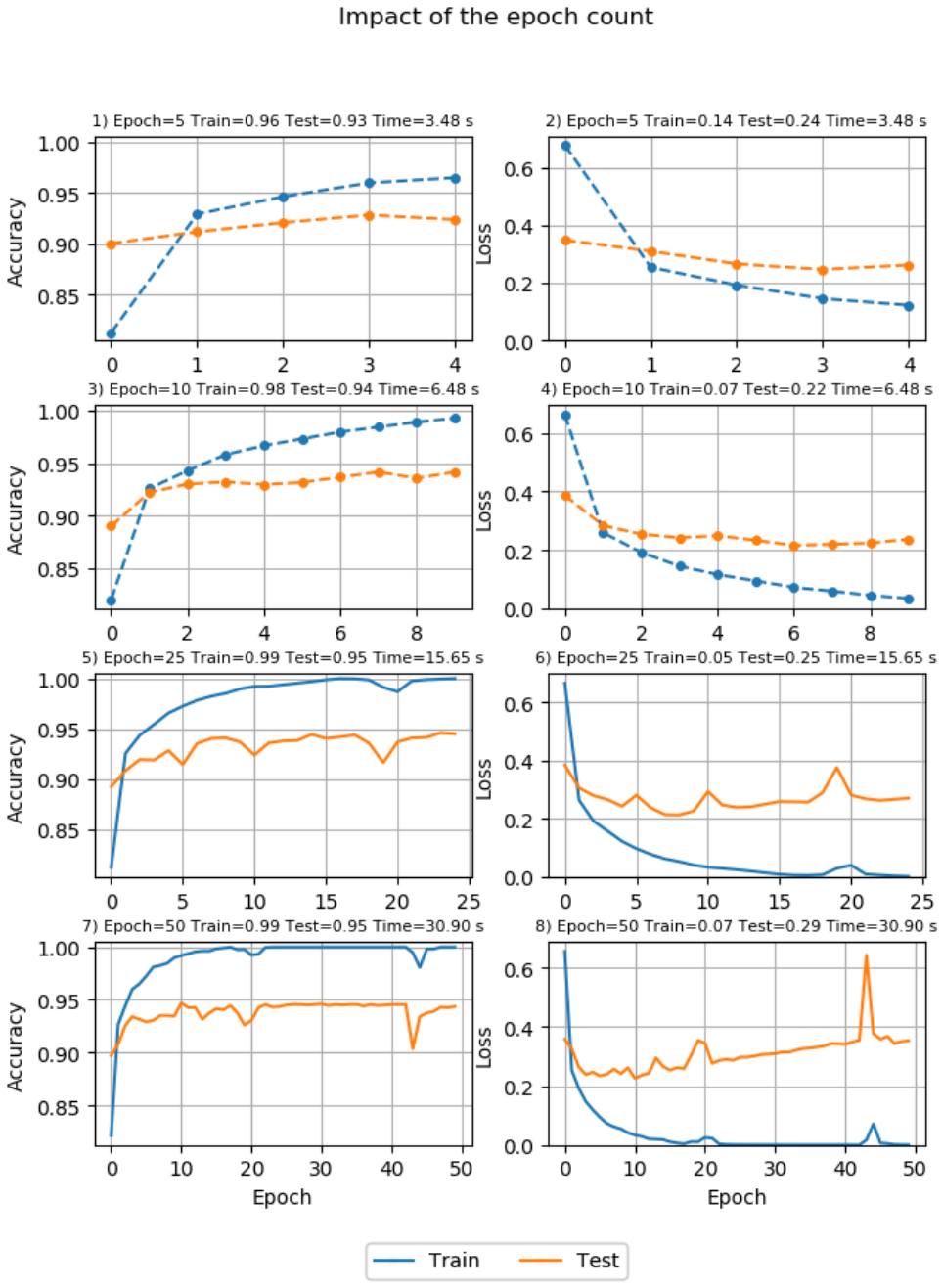
Sırasıyla aşağıdakiler incelenecektir.

- Döngü sayısının etkileri
- Yığın boyutunun etkileri
- Eğitim için kullanılan veri sayısının etkileri
- Gizli katman sayısının etkileri
- Gizli katmanlardaki nöron sayısının etkileri

Verilecek olan grafiklerde sol tarafta kalan grafikler YSA'nın başarı oranını, sağ taraftakiler ise objektif fonksiyonunun değerini göstermektedir. Mavi ile gösterilen değerler eğitim verisinin bir kısmı kullanılarak elde edilen değerlerdir, turuncu ile gösterilen değerler ise YSA'nın daha önce hiç görmediği verilerden oluşan veri seti kullanılarak elde edilmiş değerlerdir. Bu yüzden turuncu çizginin başarı oranının bire yakın olması, objektif fonksiyonu değerinin ise sıfıra yakın olması istenilen durumdur.

5.3.1 Döngü saysının (epoch) etkileri

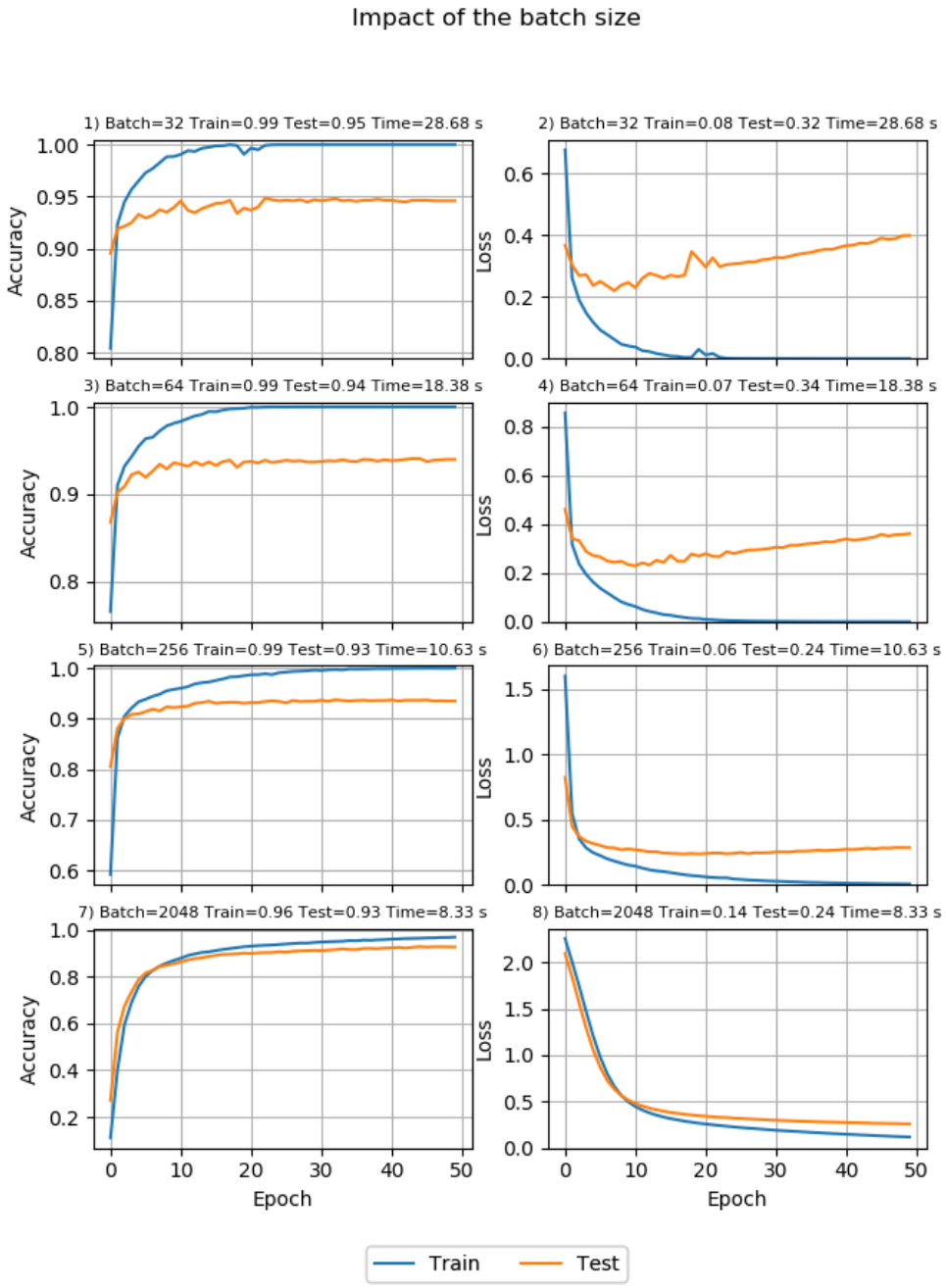
2 tane gizli, her bir katmanda 64 adet nöron, 10000 adet eğitim verisi, yığın sayısı 32 ve döngü sayısı sırasıyla 5, 10, 25, 50 seçilmiştir (Şekil 5.18). Aşağıda görüldüğü üzere döngü sayısının az seçilmesi, YSA'yı eğitmek için yeterli sayıda iterasyon yapamamaktadır ve yüksek başarı oranlarına erişememektedir. Döngü sayısının çok seçilmesi ise YSA'nın verileri ezberlemesine yol açmaktadır ve bu da YSA'nın daha önce görmediği verilerde doğru bir tahmin yapamamasına neden olmaktadır.



Şekil 5.18 : Döngü sayısına göre başarı oranı ve hata grafikleri.

5.3.2 Yığın boyutunun (batch size) etkileri

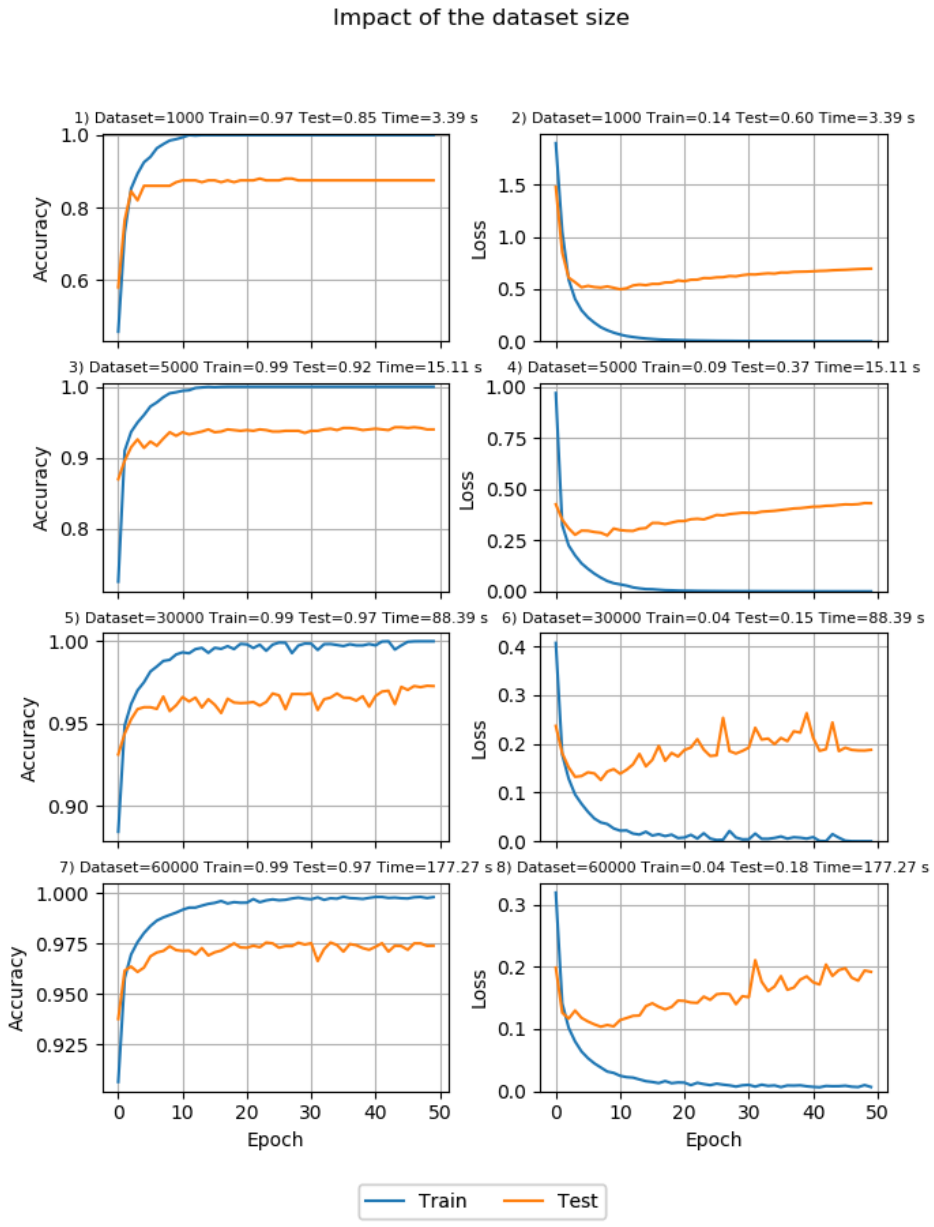
2 tane gizli, her bir katmanda 64 adet nöron, 10000 adet eğitim verisi, döngü sayısı 50 ve yığın sayısı sırasıyla 32, 64, 256, 2048 seçilmiştir (Şekil 5.19). Yığın sayısının artması, bir döngüde yapılacak olan iterasyon sayısını ters oranda etkilemektedir. Yığın sayısı küçük seçilirse, iterasyon sayısı artacaktır ve bu da YSA'nın eğitim verilerini daha çabuk ezberlemesine neden olacaktır.



Şekil 5.19 : Yığın boyutuna göre başarı oranı ve hata grafikleri.

5.3.3 Eğitim için kullanılan veri sayısının etkileri

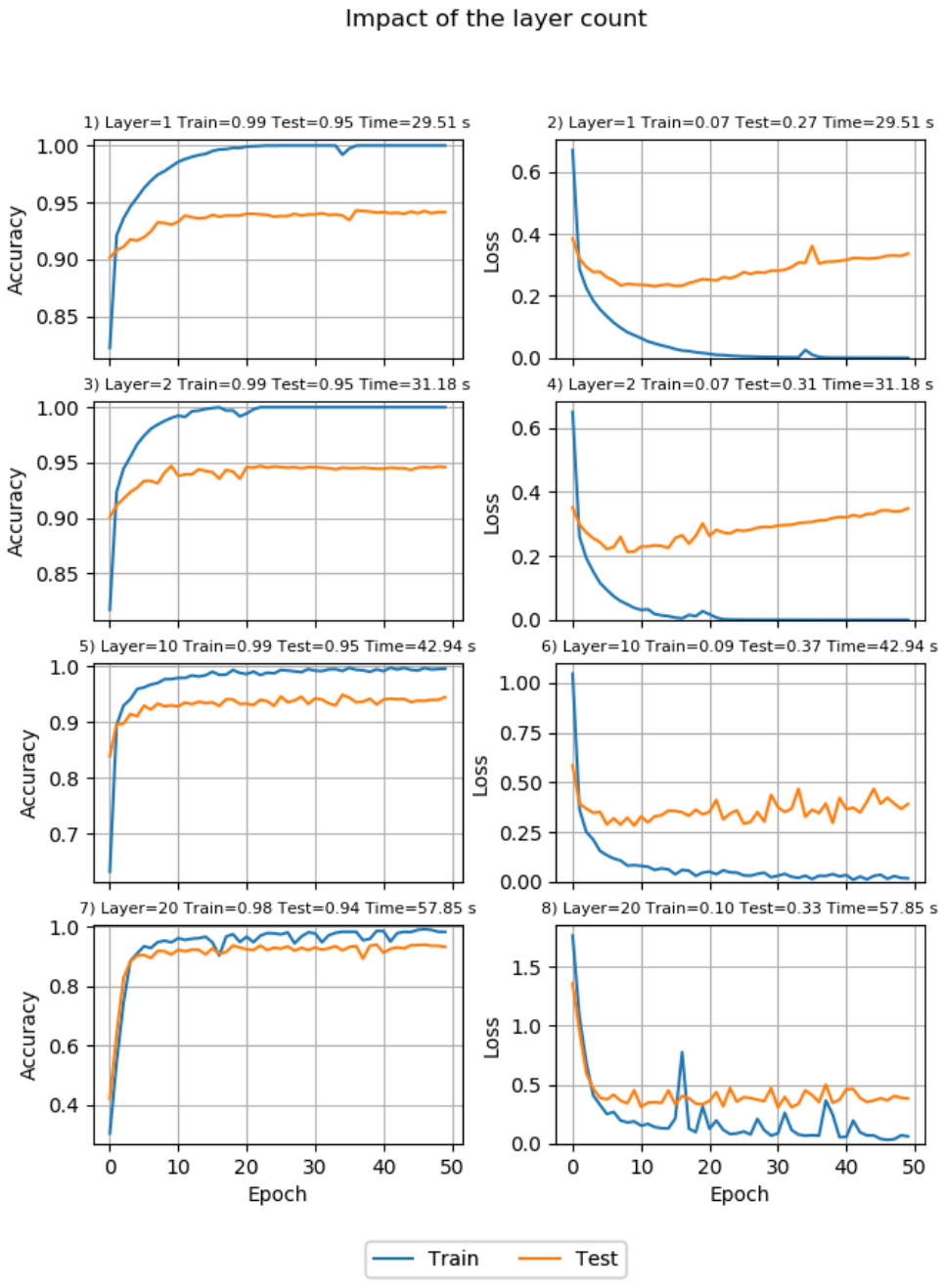
2 tane gizli, her bir katmanda 64 adet nöron, döngü sayısı 50, yığın sayısı 32 ve eğitim verisinin sayısı sırasıyla 1000, 5000, 30000, 60000 seçilmiştir (Şekil 5.20). Başarı grafiklerinden görülmektedir ki eğitim verisinin sayısı arttıkça test verilerini doğru tahmin etme oranı artmaktadır. Ayrıca eğitim verisi sayısının artışıyla, objektif fonksiyonunda elde edilen hata oranı da azalmaktadır. Ancak YSA verileri ezberleme sorunu devam etmektedir. YSA'yı eğitmek için geçen süre de veri setinin boyutu ile artmaktadır.



Şekil 5.20 : Veri setinin boyutuna göre başarı oranı ve hata grafikleri.

5.3.4 Gizli katman sayısının etkileri

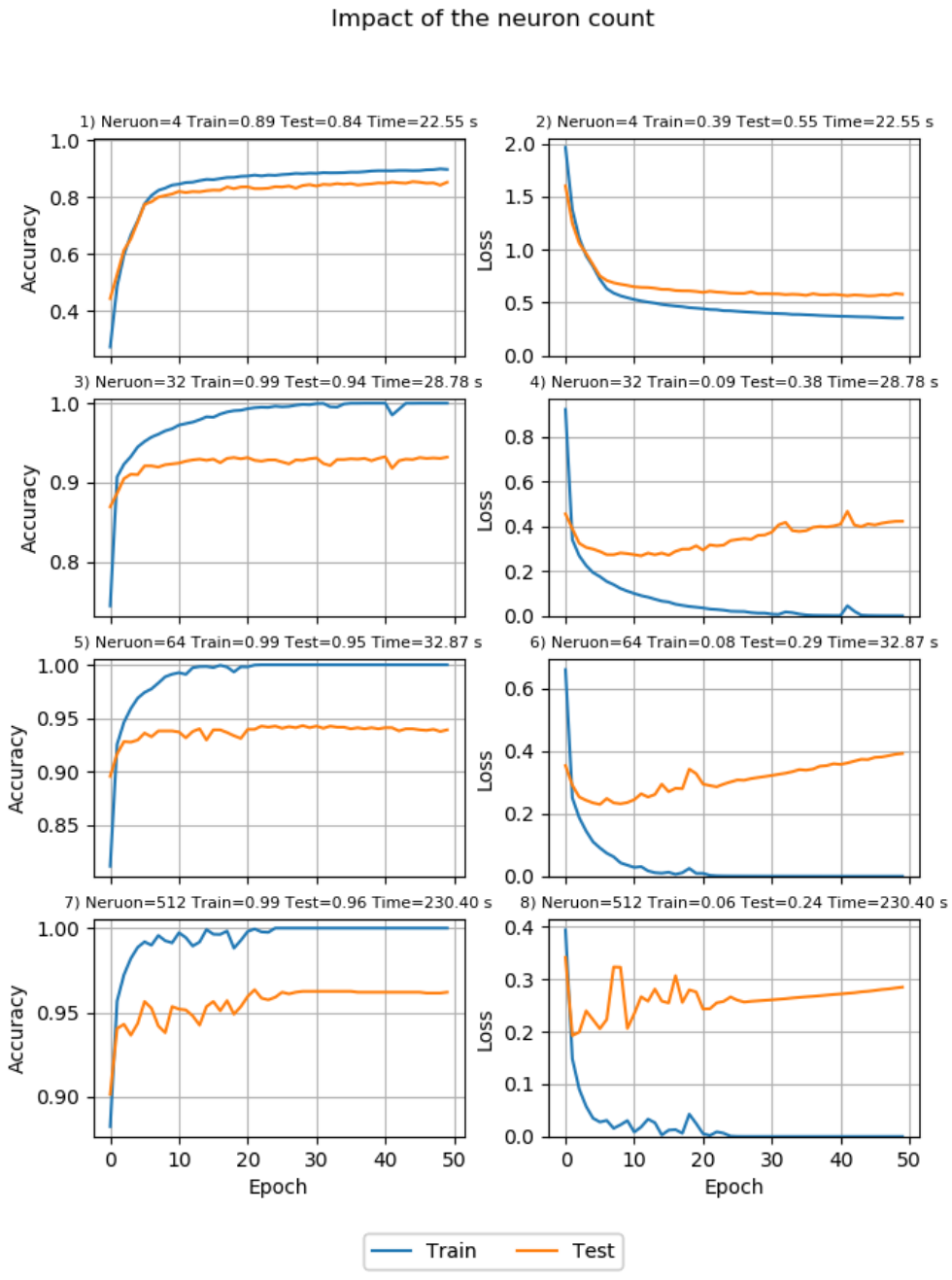
Her bir katmanda 64 adet nöron, döngü sayısı 50, yığın sayısı 32, eğitim verisinin sayısı 10000, gizli katman sayısı ise sırasıyla 1, 2, 10, 20 seçilmiştir (Şekil 5.21). Grafikte görüldüğü üzere gizli katman sayısı arttıkça ezberleme azalmaktadır. 20 tane gizli katmanın olduğu grafikte, başarı oranı her ne kadar düşük başlasa da iyi bir noktaya gelmektedir. Gizli katmanın artışıyla güncellenecek olan parametre sayısı da arttığından, YSA'yı eğitme süresi de artış göstermiştir.



Şekil 5.21 : Gizli katman sayısına göre başarı oranı ve hata grafikleri.

5.3.5 Gizli katmandaki nöron sayısının etkileri

Gizli katman sayısı 2, döngü sayısı 50, yığın sayısı 32, eğitim verisinin sayısı 10000, gizli katmanlardaki nöron sayısı ise sırasıyla 4, 32, 64, 512 seçilmiştir (Şekil 5.22). Grafikte 4 tane nöron seçildiği duruma bakılırsa, eğitim verisine ait hata değeri sıfıra ulaşmamaktadır. Bu açık bir şekilde daha fazla nöron eklenmesi gerektiğinin göstergesidir. Nöron sayısı arttıkça başarı oranı yükselmekte, hata azalmaktadır; ancak YSA'nın ezberleme problemi devam etmektedir.



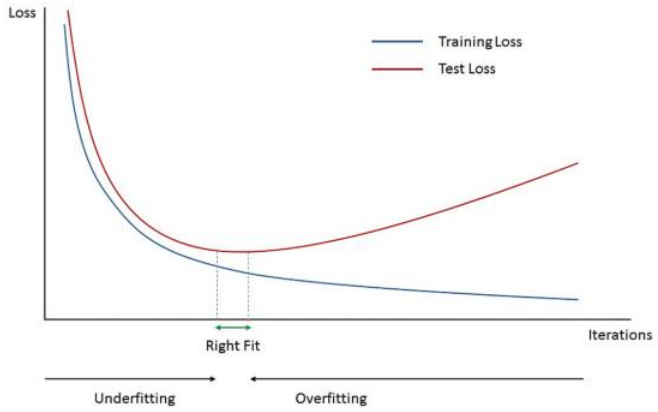
Şekil 5.22 : Gizli katmandaki nöron sayısına göre başarı oranı ve hata grafikleri.

5.4 Ezberlemenin önüne geçmenin yolları

Şekil 5.23'te ezberlemeye başladığı nokta açıkça gözükmektedir. Önüne geçmek için bazı yöntemler kullanılabilir. Bunlardan bazıları aşağıdaki gibidir.

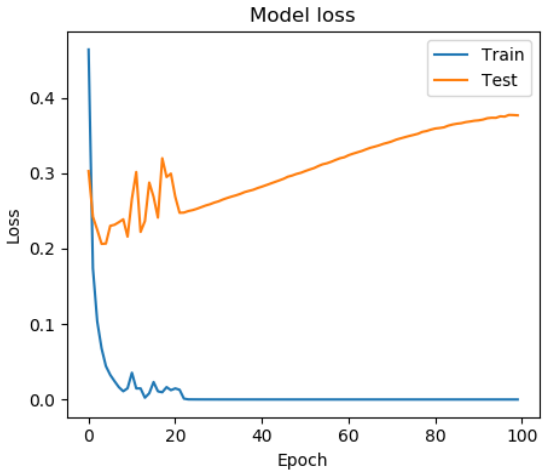
- Ezberlemenin başladığı noktayı algılayıp eğitimi erken bitirme
- L1, L2 regülasyonu kullanma
- Seyreltme (dropout) uygulama

Bu uygulamada L1 ve L2 regülasyon beraber kullanılacaktır.



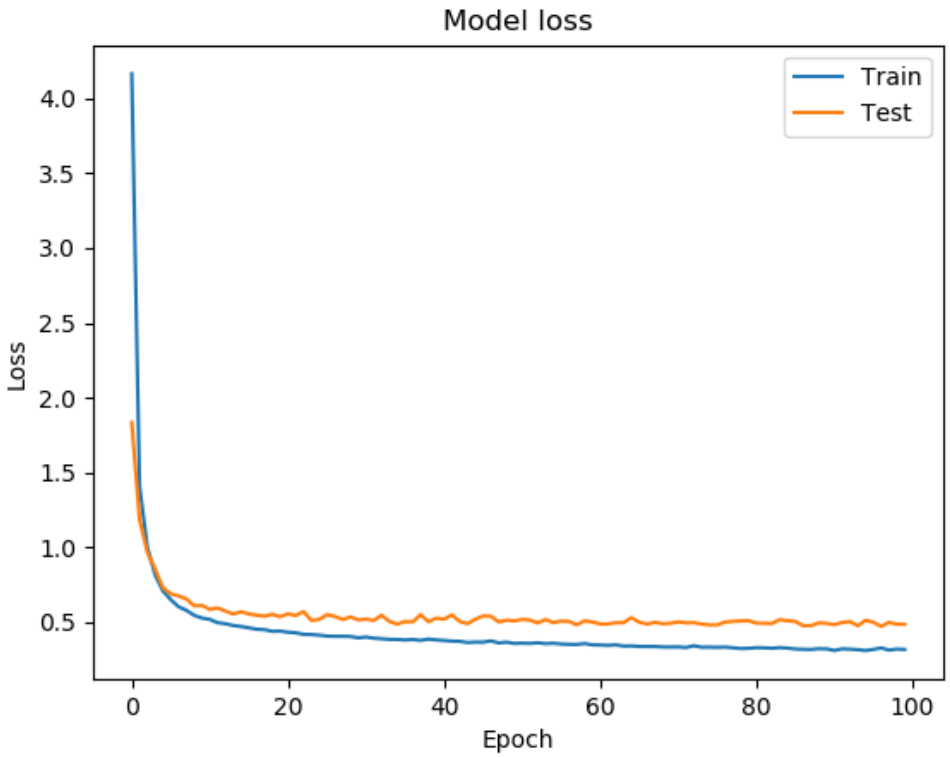
Şekil 5.23 : Eğitim ve test verisinin hata grafiği.

Şekil 5.24'te 2 tane gizli katmanı olan ve her bir katmanında 256 adet nöron bulunan bir modele ait hata grafiği verilmiştir. Test verisi ile hesaplanan hata değerinin sürekli bir şekilde arttığı gözükmektedir. Bunun önüne geçmek için, her bir katmana L1 ve L2 katsayıları verilecektir.



Şekil 5.24 : Ezberleme yapmış YSA'ya ait hata grafiği.

Şekil 5.25'de görüldüğü üzere, ezberlemenin önüne geçilmiştir.



Şekil 5.25 : L1=0.001 ve L2=0.0001 kullanılan YSA'ya ait hata grafiği.

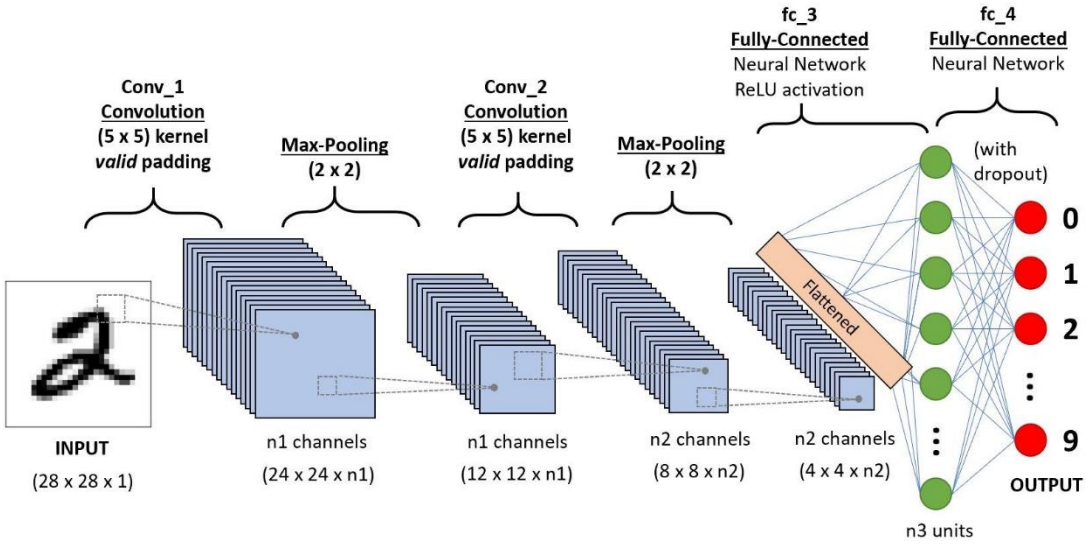
5.5 Evrişimsel Yapay Sinir Ağı

Öncelikle Python kodunda eğitim ve test verileri değişkenlere atanmalıdır (Şekil 5.26). Ayrıca evrişimsel yapay sinir ağı kullanılacağından, yapay sinir ağının giriş boyutlarından değişiklik yapılmak zorundadır.

```
# Verilerin yüklenmesi
(training_data, training_labels), (test_data, test_labels) = tf.keras.datasets.mnist.load_data()
training_data, test_data = (training_data) / 255, (test_data) / 255
# Verileri yeniden şekillendirme [örnek sayısı][genişlik][uzunluk][kanal sayısı]
training_data = training_data.reshape((training_data.shape[0], 28, 28, 1)).astype('float32')
test_data = test_data.reshape((test_data.shape[0], 28, 28, 1)).astype('float32')
```

Şekil 5.26 : El yazısı verilerinin değişkenlere atanması.

Yapılacak olan işlem, görüntüleri kategorize etmek olduğu için evrişimsel yapay sinir ağı kullanılacaktır (Şekil 5.27). Uygulamanın çeşidine göre yapay sinir ağı modeli tasarlanmaktadır ve evrişimsel yapay sinir ağları sınıflama alanında başarılı sonuç vermektedir.



Şekil 5.27 : Evrişimsel yapay sinir ağı.

28'e 28 pikselden oluşmaktadır her bir resim. Bu resimler üzerinde evrişim (convolution) işlemi uygulanır ve görüntünün özellikleri çıkartılır. Daha sonra havuzlama işlemi uygulanır ve baskın özellikler filtrelenir. Benzer işlemler bir kere daha yapıldıktan sonra elde edilen dizi düzleştirilir yani bir boyutlu hale getirilir ve yapay sinir ağına beslenir. Sonrasında kullanılacak olan gizli katman sayısı ve bu katmanlardaki nöron sayısı, deneme yanılma ile bulunabilir. Şekilde yapay sinir ağı modelinin Python kodunda oluşturulmuş hali görülmektedir (Şekil 5.28).

```
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), input_shape=(28, 28, 1),activation="relu", padding="same"),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3),activation="relu", padding="same"),
    tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
    # tf.keras.layers.Dropout(0.2), #aşırı uyum gösterme olduğunda kullanılabilir
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(256, activation=tf.nn.relu),
    tf.keras.layers.Dense(256, activation=tf.nn.relu),
    tf.keras.layers.Dense(10, activation=tf.nn.softmax)
])
```

Şekil 5.28 : Evrişimsel yapay sinir ağı modeli.

Model tanımlandıktan sonra bu modeli derlemek ve eğitmek gerekmektedir ve bu işlemler aşağıdaki gibi yapılabilir (Şekil 5.29).

```
# Modelin derlenmesi
model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam', metrics=['accuracy'])
# Modelin eğitilmesi
history = model.fit(training_data, training_labels, batch_size=1000, epochs=5, validation_split=0.1, verbose=2)
```

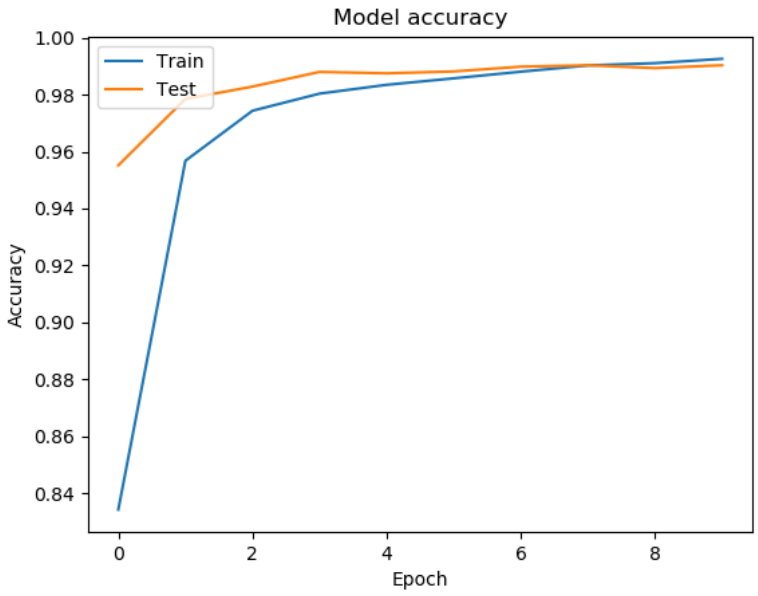
Şekil 5.29 : Yapay sinir ağının derlenmesi ve eğitilmesi.

Eğitimin adımları ve bu adımlarda elde edilen sonuçlar aşağıdaki gibidir (Şekil 5.30).

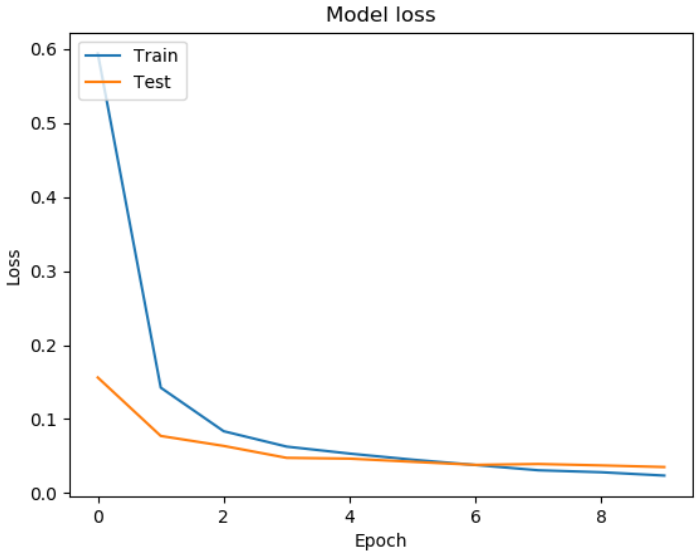
```
Train on 54000 samples, validate on 6000 samples
Epoch 1/10
 - 102s - loss: 0.5937 - acc: 0.8342 - val_loss: 0.1562 - val_acc: 0.9552
Epoch 2/10
 - 102s - loss: 0.1425 - acc: 0.9568 - val_loss: 0.0773 - val_acc: 0.9785
Epoch 3/10
 - 102s - loss: 0.0837 - acc: 0.9744 - val_loss: 0.0638 - val_acc: 0.9828
Epoch 4/10
 - 102s - loss: 0.0629 - acc: 0.9804 - val_loss: 0.0477 - val_acc: 0.9880
Epoch 5/10
 - 102s - loss: 0.0535 - acc: 0.9835 - val_loss: 0.0467 - val_acc: 0.9875
Epoch 6/10
 - 102s - loss: 0.0453 - acc: 0.9857 - val_loss: 0.0424 - val_acc: 0.9882
Epoch 7/10
 - 102s - loss: 0.0381 - acc: 0.9881 - val_loss: 0.0383 - val_acc: 0.9898
Epoch 8/10
 - 102s - loss: 0.0309 - acc: 0.9903 - val_loss: 0.0394 - val_acc: 0.9903
Epoch 9/10
 - 102s - loss: 0.0283 - acc: 0.9911 - val_loss: 0.0375 - val_acc: 0.9893
Epoch 10/10
 - 103s - loss: 0.0238 - acc: 0.9926 - val_loss: 0.0353 - val_acc: 0.9903
```

Şekil 5.30 : Eğitim adımları.

Elde edilen model çok iyi bir doğruluk oranına (Şekil 5.31) ve düşük hataya sahiptir (Şekil 5.32). Uygulamada bu model kullanılacaktır. Eğitilmiş olan model daha sonra tekrar kullanılmak üzere kayıt edilir ve Python programının tekrar açılmasıyla kayıt edilen model yüklenir. Bu sayede modeli tekrar eğitme adımı atlanır ve eğitilmesi beklenmemiş olur.



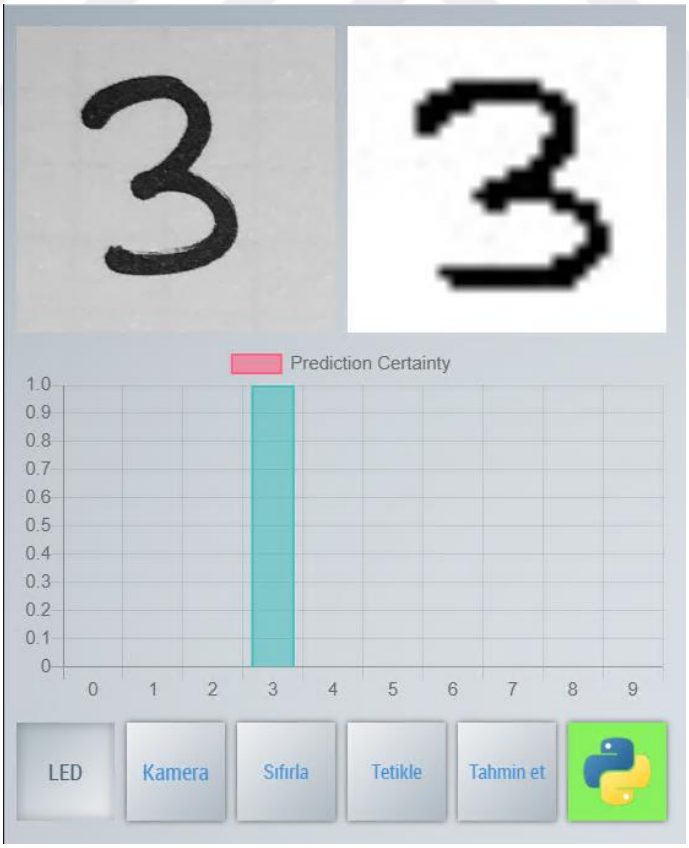
Şekil 5.31 : Modelin başarı grafiği.



Şekil 5.32 : Modelin hata grafiği.

5.6 Arayüz Tasarımı

Şekil 5.33'te TwinCAT HMI ile tasarlanmış arayüz sayfası verilmiştir (Şekil 5.33).



Şekil 5.33 : TwinCAT HMI sayfası.

Sayfada sol üst tarafta kalan görüntü, endüstriyel kameradan veya dosya kaynağından gelen ham ve işlenmemiş görüntüdür. Sağ üst tarafta kalan görüntü ise Python koduna gönderilmeden önce TwinCAT Vision algoritmaları kullanılarak üzerinde işlem yapılmış görüntüdür.

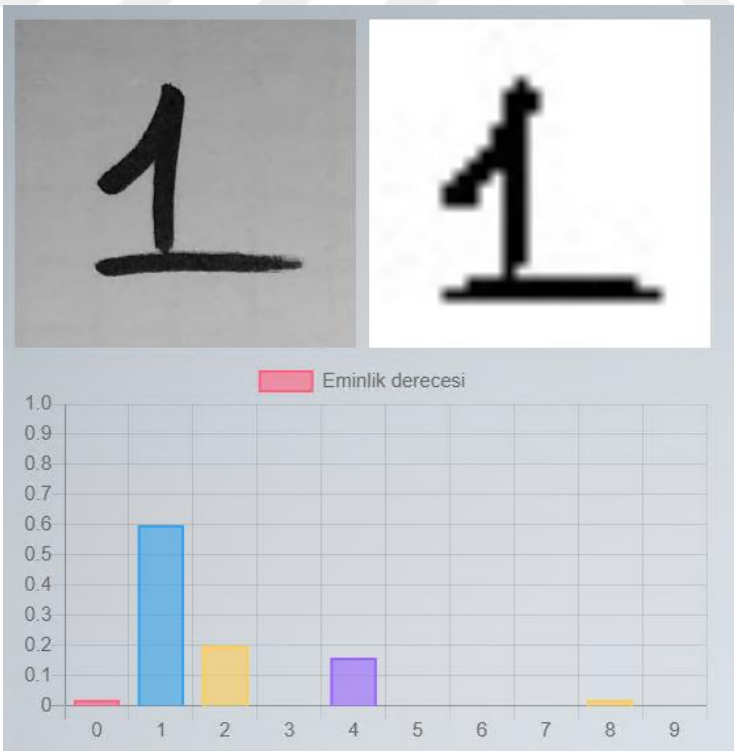
Orta tarafta kalan sütun grafiği ise yapay sinir ağının çıkışlarını göstermektedir. Şekil 5.33'te verilmiş olan grafiğe bakıldığında, yapay sinir ağı tarafından yapılmış olan tahminin 1 yani %100 oranında 3 olarak sınıflandırdığı görülmektedir.

Bazı durumlarda da doğru tahmin etmiş olsa bile eminlik derecesi yüksek olamayabiliyor (Şekil 5.34). Şekilde yazılı olan 1 rakamı aslında yazım olarak yanlış yazılmıştır ve bu yüzden de eminlik derecesi düşük çıkmıştır.

"Tahmin et" butonuna basıldıktan sonra 28x28 bayt dizisi Python koduna gönderilmektedir ve yapay sinir ağının sonuçları da PLC'ye gönderilmektedir.

Eğer kamera aktif değilse "Tetikle" butonu ile çevrimdışı dosyalar kullanılabilir ve butona her basışta, resim değişmektedir.

"LED" butonu eğer giriş çıkış modüllerine bağlı bir ışıklandırma varsa bu ışıklandırmayı açıp kapatmaya yaramaktadır.



Şekil 5.34 : TwinCAT HMI sayfası.

6. SONUÇ VE ÖNERİLER

Bu proje çalışmasında endüstriyel ekipmanlar kullanılarak bir görüntü işleme uygulaması yapılmıştır. Windows işletim sistemine sahip bir endüstriyel bilgisayar kullanılarak yüksek çözünürlükte görüntü alma olağanı sağlayan ve zorlu çevre şartlarında da çalışabilecek bir endüstriyel kamera ile haberleşme yapılmıştır. Kameradan alınan ve el yazısıyla yazılmış tekil rakamları içeren görüntüler, TwinCAT otomasyon yazılım ortamında bazı ön işlemlere tabi tutulduktan sonra Python dilinde yazılmış ve Tensorflow kütüphanesi kullanarak hazırlanmış evrişimsel yapay sinir ağı tarafından sınıflandırılmaktadır. Sınıflandırmanın sonucu da kullanıcıya TwinCAT HMI arayüzü ile sunulmaktadır.

Kullanılan endüstriyel kamera GigE Vision standardını desteklemektedir ve TwinCAT Vision eklentisi, GigE Vision destekleyen herhangi bir endüstriyel kamera ile rahatlıkla haberleşmektedir. Kameranın olmadığı durumlarda ise harici olarak projeye eklenen görüntüler kullanılabilir.

TwinCAT Vision eklentisi ile gelen PLC kütüphanesinde görüntü işleme algoritmaları bulunmaktadır ve hiçbir şekilde PLC ortamından ayrılmadan görüntü işleme yapmak mümkündür. Bu uygulamada da alınan görüntü yapay sinir ağına aktarılmadan önce PLC koduyla üzerinde görüntü işleme yapılmaktadır.

Yapay sinir ağını eğitmek için Tensorflow içerisinde gelen, 28x28 pikselden oluşan ve el yazısı ile yazılmış 70000 adet resim bulunmaktadır. Bu resimlerden 54000 tanesi yapay sinir ağını eğitmek, 6000 tanesi modelin eğitimi sırasındaki döngülerde test yapmak için, 10000 tanesi de model tamamen eğitildikten sonra test yapmak için kullanılmaktadır ve bu test verileri, modelin daha önce hiç görmemiş olduğu verilerdir. Eğitilen model %99 başarı oranına sahiptir.

Kullanılacak olan yapay sinir ağı seçilirken yapılacak olan uygulama göz önüne alınmalıdır. En çok öne çıkan çeşitler şu şekilde verilebilir; tekil algılayıcı, ileri beslemeli sinir ağı, derin ileri beslemeli sinir ağı, tekrarlayan sinir ağı, uzun/kısa süreli bellek, Markov zinciri, Boltzman makinesi, Hopfield ağı, destek vektör makinesi,

evrişimsel sinir ağı, derin inanç ağacı. Her birinin kullanım alanı farklıdır. Örneğin; tekrarlayan yapay sinir ağı modelini kullanarak, giriş verilerine göre bir tahmin üreten uygulama oluşturmak mümkündür.

Proje için evrişimsel yapay sinir ağı kullanılmıştır çünkü sınıflandırma uygulaması yapılacaktır; ancak nöron sayısı veya gizli katman sayısı gibi sinir ağı modellerini oluşturan bazı seçimlerin veya modeli eğitirken kullanılan döngü sayısı, yığın sayısı gibi bazı değişkenlerin, modelin başarısı üzerine etkileri tartışılmış, grafikler ile karşılaştırmalar yapılmıştır. Yapay sinir ağlarında çokça karşılaşılan ezberleme sorunu örnekler ile gösterilmiş, bazı çözüm önerileri sunulmuştur.

Son olarak, kullanıcının interaktif olabilmesi ve rahatlıkla süreci takip edebilmesi için bir arayüz geliştirilmiştir ve bu arayüz endüstriyel bilgisayar ile aynı alt ağda olan herhangi bir cihazdan erişilebilen bir arayüzdür. Bilgisayar, tablet veya telefon gibi cihazlardan erişilebilir.

KAYNAKLAR

[1] **Ji, K. ve Song, Z.** (2012). System and method for verification and validation of redundancy software in plc systems, uS Patent App. 13/415,897.

[2] **Attigeri, S.** (2018). Neural Network Based Handwritten Character Recognition System.

[3] **Hecht-Nielsen, R.** (1988). Neurocomputing: picking the human brain. IEEE spectrum, 25(3), 36-41.

[4] **Url-1** <*https://tr.wikipedia.org/wiki/Bilgisayar_tarihi*>, erişim tarihi 03.03.2020.

[5] **Gustavson, T.** (2009). Camera: a history of photography from daguerreotype to digital. Sterling Innovation.

[6] **Young, H. D. ve Freedman, R. A.** (2008). Sears and Zemansky's university physics (Vol. 1).

[7] **London, B. ve Upton, J. ve Kobré, K. ve Brill, B.** (2002). Photography (7 ed.). Upper Saddle River, New Jersey: Prentice Hall. ISBN 978-0-13-028271-2.

[8] **Url-2** <*https://www.cambridgeincolour.com/tutorials/camera-lenses.htm*>, erişim tarihi 15.03.2020.

[9] **Url-3** <*https://electronics.howstuffworks.com/autofocus.htm*>, erişim tarihi 15.03.2020.

[10] **Url-4** <*https://pdfs.semanticscholar.org/5049/a5c0364509c19b2c64c220645096d52d212d.pdf*>, erişim tarihi 20.03.2020.

[11] **Fischler, M. A. ve Firschein, O.** (1987). Intelligence: The eye, the brain, and the computer.

[12] **Noakes, P**. Introduction to Biological Neural Systems, Technical Report, Dept. of Electronic Systems Eng., University of Essex, Colchester, Essex, CO4 3SQ.

[13] **Müller, B. ve Reinhardt, J.** (1990). Neural Networks: An Introduction, Springer-Verlag.

[14] **Kandel, E.C., Schwartz, J.H.** (1985). Principles of Neural Science, Elsevier, Amsterdam.

[15] **Sejnowski, J. T.** (1981). Skeleton fields in the brain. Parallel Models of Associative Memory.

[16] **McCulloch, W. S. ve Pitts, W.** (1943). A logical calculus of the ideas immanent in nervous activity. The bulletin of mathematical biophysics, 5(4), 115-133.

EKLER

EK A: YSA modeli, modelin eğitimi, ADS haberleşmesi (digit_recognizer.py)

EK B: YSA testleri (impact_tests.py)

EK C: TwinCAT PLC projesi

EK A

```
# Kullanilan kutuphaneler
import sys
import os
import tensorflow as tf
import numpy as np
import pyads
import matplotlib.pyplot as plt
from ctypes import sizeof
import time
import threading
import array

# Global degiskenler
stop_threads = False
plc_con_status = False
is_convolutional = True

# Verilerin yüklenmesi
(training_data, training_labels), (test_data, test_labels) =
tf.keras.datasets.mnist.load_data()
training_data, test_data = (training_data) / 255, (test_data) / 255
# Verileri yeniden şekillendirme [örnek sayısı][genişlik][uzunluk][kanal
sayısı]
if is_convolutional:
    training_data = training_data.reshape((training_data.shape[0], 28, 28,
1)).astype('float32')
    test_data = test_data.reshape((test_data.shape[0], 28, 28,
1)).astype('float32')

# PLC ile haberleşme objesi oluşturma
plc = pyads.Connection('127.0.0.1.1.1', 851)
# PLC haberleşmesinin açılması
plc.open()

if not os.path.isfile('C:\\Tez\\model.h5'):
    print("Model bulunamadi, model bastan egitiliyor!")
    # model yok, eğit ve modeli kaydet
    if is_convolutional:
        model = tf.keras.Sequential([
            tf.keras.layers.Conv2D(32, (3, 3), input_shape=(28, 28,
1),activation="relu", padding="same"),
            tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
            tf.keras.layers.Conv2D(64, (3, 3),activation="relu",
padding="same"),
            tf.keras.layers.MaxPool2D(pool_size=(2, 2)),
            #tf.keras.layers.Dropout(0.2), # Ezberleme olursa bu katman
kullanilabilir
            tf.keras.layers.Flatten(),
            tf.keras.layers.Dense(256, activation=tf.nn.relu),
            tf.keras.layers.Dense(256, activation=tf.nn.relu),
            tf.keras.layers.Dense(10, activation=tf.nn.softmax)
        ])
    else:
        model = tf.keras.Sequential([
            tf.keras.layers.Flatten(input_shape=(28,28)),
            tf.keras.layers.Dense(256, activation=tf.nn.relu,
kernel_regularizer= tf.keras.regularizers.l1_l2(l1=0.001, l2=0.0001)),
            tf.keras.layers.Dense(256, activation=tf.nn.relu,
kernel_regularizer= tf.keras.regularizers.l1_l2(l1=0.001, l2=0.0001)),
            tf.keras.layers.Dense(10, activation=tf.nn.softmax)
```

```
        ])

    # Modelin ozeti
    model.summary()
    # Modelin derlenmesi
    model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam',
metrics=['accuracy'])
    # Modelin eğitilmesi
    history = model.fit(training_data, training_labels, batch_size=1000,
epochs=10, validation_split=0.1, verbose=2)
    # Modelin test edilmesi
    train_acc = model.evaluate(training_data, training_labels)
    test_acc = model.evaluate(test_data, test_labels)

    # Basari grafigi
    plt.plot(history.history['acc'])
    plt.plot(history.history['val_acc'])
    plt.title('Model accuracy')
    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc = 'lower right')
    plt.show()
    # Hata grafigi
    plt.plot(history.history['loss'])
    plt.plot(history.history['val_loss'])
    plt.title('Model loss')
    plt.ylabel('Loss')
    plt.xlabel('Epoch')
    plt.legend(['Train', 'Test'], loc = 'upper right')
    plt.show()

    # Egitilen modelin kayit edilmesi
    model.save("C:\\Tez\\model.h5")
else:
    print("Model bulundu!")
    # Eger model varsa hazır, onu kullan
    model = tf.keras.models.load_model('C:\\Tez\\model.h5')

def GetInformation(var):
    print("Type :", type(var))
    print("Size :", sys.getsizeof(var))

graph = tf.get_default_graph()

# PLC tarafindan cagirilacak olan tahmin etme komutu
def predictCommand():
    arrFromPlc = plc.read_by_name('GVL.stPictureEx', pyads.PLCTYPE_INT * 28 *
28)
    testArray = np.expand_dims((255 - np.array(arrFromPlc))/255.,0)
    if is_convolutional:
        testArray = testArray.reshape((1, 28, 28, 1)).astype('float32')
    GetInformation(testArray)
    with graph.as_default():
        plcPredictions = model.predict(testArray)
    plc.write_by_name('GVL.arrPredictions', plcPredictions.tolist()[0],
pyads.PLCTYPE_REAL * 10)

# PLC baglantisini kontrol eden islem parcasi
def connection_check_thread(name):
    global plc_con_status
    while True:
        if stop_threads:
```

```
            break
        try:
            plc.read_state()
            if plc_con_status == False:
                plc.read_by_name('GVL.nCounter', pyads.PLCTYPE_INT)
                print('CONNECTED TO PLC!')
                plc_con_status = True
            plc.write_by_name('GVL.nPyCounter',
plc.read_by_name('GVL.nCounter', pyads.PLCTYPE_INT),
plc_datatype=pyads.PLCTYPE_INT)
        except pyads.ADSError as ex:
            if plc_con_status == True:
                print('CONNECTION ERROR. WAITING FOR RECONNECTION!')
                plc_con_status = False
        time.sleep(1)

# PLC'den gelecek olan tahmin et komutunu kontrol eden islem parcasi
def predict_command_thread(name):
    while True:
        if stop_threads:
            break
        if plc_con_status == True:
            try:
                if  plc.read_by_name('GVL.bPredict', pyads.PLCTYPE_BOOL):
                    predictCommand()
                    plc.write_by_name('GVL.bPredict', False ,
pyads.PLCTYPE_BOOL)
            except pyads.ADSError as ex:
                print(ex.msg)
        time.sleep(0.1)

# islem parcalari
conCheck = threading.Thread(target=connection_check_thread, args=(1,))
cyclicRead = threading.Thread(target=predict_command_thread, args=(1,))
conCheck.start()
cyclicRead.start()
input("Cikmak icin herhangi bir tusa basin!")
# islem parcalarini kapatir ve programi sonlaririr
stop_threads = True
conCheck.join()
cyclicRead.join()
################################################################################
```

EK B

```
# Kullanilan kutuphaneler
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
import time

# Verilerin yuklenmesi
(train_x, train_y), (test_x, test_y) = tf.keras.datasets.mnist.load_data()
train_x, test_x = (train_x) / 255, (test_x) / 255

def fit_model(model, trainX, trainy, testX, testy, n_batch, n_epoch, n_figure,
title):
       # modeli derle
    model.compile(loss='sparse_categorical_crossentropy', optimizer='Adam',
metrics=['accuracy'])
       # modeli egit
    start_time = time.time()
    history = model.fit(trainX, trainy, batch_size=n_batch, epochs=n_epoch,
validation_split=0.2, verbose=0)
    execution_time = time.time() - start_time
    # modeli test et
    train_acc = model.evaluate(trainX, trainy,  verbose=0)
    test_acc = model.evaluate(testX, testy,  verbose=0)
    print('Batch Size : %d, Train: %.3f, Test: %.3f Time=%.2f s' % (n_batch,
train_acc[1], test_acc[1], execution_time))

    # Basari grafigini ciz
    fig = plt.figure(n_figure)
    fig.suptitle('Impact of the ' + title)
    plot_no = 420 + (2*i+1)
    ax = plt.subplot(plot_no)
    if n_epoch >= 20:
        plt.plot(history.history['acc'], label='Train')
        plt.plot(history.history['val_acc'], label='Test')
    else :
        plt.plot(history.history['acc'], label='Train' , marker='o',
linestyle='dashed', markersize=4)
        plt.plot(history.history['val_acc'], label='Test', marker='o',
linestyle='dashed', markersize=4)

    if title == 'batch size':
        plt.title('%d) Batch=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+1),
n_batch, train_acc[1], test_acc[1], execution_time), fontsize=8)
    elif title == 'dataset size':
        plt.title('%d) Dataset=%d Train=%.2f Test=%.2f Time=%.2f s' %
((2*i+1), len(trainX), train_acc[1], test_acc[1], execution_time), fontsize=8)
    elif title == 'epoch count':
        plt.title('%d) Epoch=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+1),
n_epoch, train_acc[1], test_acc[1], execution_time), fontsize=8)
    elif title == 'layer count':
        plt.title('%d) Layer=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+1),
layer_count[i], train_acc[1], test_acc[1], execution_time), fontsize=8)
    elif title == 'neuron count':
        plt.title('%d) Neruon=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+1),
neuron_count[i], train_acc[1], test_acc[1], execution_time), fontsize=8)

    plt.ylabel('Accuracy')
    if (i == 3):
        plt.xlabel('Epoch')
    else:
```

```
        if title != 'epoch count':
            ax.axes.xaxis.set_ticklabels([])
    plt.grid(True)
    ax.set_ylim(top=1.005)

    # Hata grafigini ciz
    plot_no = 420 + (2*i+2)
    ax = plt.subplot(plot_no)
    if n_epoch > 20:
        plt.plot(history.history['loss'], label='Train')
        plt.plot(history.history['val_loss'], label='Test')
    else:
        plt.plot(history.history['loss'], label='Train', marker='o',
linestyle='dashed', markersize=4)
        plt.plot(history.history['val_loss'], label='Test', marker='o',
linestyle='dashed', markersize=4)

    if title == 'batch size':
        plt.title('%d) Batch=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+2),
n_batch, train_acc[0], test_acc[0], execution_time), fontsize=8)
    elif title == 'dataset size':
        plt.title('%d) Dataset=%d Train=%.2f Test=%.2f Time=%.2f s' %
((2*i+2), len(trainX), train_acc[0], test_acc[0], execution_time), fontsize=8)
    elif title == 'epoch count':
        plt.title('%d) Epoch=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+2),
n_epoch, train_acc[0], test_acc[0], execution_time), fontsize=8)
    elif title == 'layer count':
        plt.title('%d) Layer=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+2),
layer_count[i], train_acc[0], test_acc[0], execution_time), fontsize=8)
    elif title == 'neuron count':
        plt.title('%d) Neruon=%d Train=%.2f Test=%.2f Time=%.2f s' % ((2*i+2),
neuron_count[i], train_acc[0], test_acc[0], execution_time), fontsize=8)

    plt.ylabel('Loss')
    if (i == 3):
        plt.xlabel('Epoch')
    else:
        if title != 'epoch count':
            ax.axes.xaxis.set_ticklabels([])
    plt.grid(True)
    handles, labels = ax.get_legend_handles_labels()
    plt.figlegend( handles, labels, loc = 'lower center', ncol=2)
    ax.set_ylim(bottom=0)

# Modeli tanimla
def create_model(layer_count=2, neuron_count=64):
    model = tf.keras.Sequential()
    model.add(tf.keras.layers.Flatten(input_shape=(28,28)))
    for i in range(layer_count):
        model.add(tf.keras.layers.Dense(neuron_count, activation=tf.nn.relu))
    model.add(tf.keras.layers.Dense(10, activation=tf.nn.softmax))

    return model

batch_sizes = [32, 64, 256, 2048]
# Impact of the batch size
for i in range(len(batch_sizes)):
    fit_model(create_model(), train_x[:10000], train_y[:10000], test_x[:1000],
test_y[:1000], batch_sizes[i], n_epoch=50, n_figure=1, title='batch size')

dataset_sizes = [1000, 5000, 30000, 60000]
# Impact of the dataset size
```

```
for i in range(len(dataset_sizes)):
    fit_model(create_model(), train_x[:dataset_sizes[i]],
train_y[:dataset_sizes[i]], test_x[:1000], test_y[:1000], n_batch=32,
n_epoch=50, n_figure=2, title='dataset size')

epoch_count = [5,10,25,50]
# Impact of the epoch count
for i in range(len(epoch_count)):
    fit_model(create_model(), train_x[:10000], train_y[:10000], test_x,
test_y, n_batch=32, n_epoch=epoch_count[i], n_figure=3, title='epoch count')

layer_count = [1,2,10,20]
# Impact of the layer count
for i in range(len(layer_count)):
    fit_model(create_model(layer_count=layer_count[i]), train_x[:10000],
train_y[:10000], test_x, test_y, n_batch=32, n_epoch=50, n_figure=4,
title='layer count')

neuron_count = [4,32,64,512]
# Impact of the neuron count
for i in range(len(neuron_count)):
    fit_model(create_model(neuron_count=neuron_count[i]), train_x[:10000],
train_y[:10000], test_x, test_y, n_batch=32, n_epoch=50, n_figure=5,
title='neuron count')

plt.show()
################################################################################
```

EK C

```
VAR
	fbFileSource		: FB_VN_GevCameraControl;
	eState			: ETcVnCameraState;
	hr				: HRESULT;
	// Images
	ipImageIn			: ITcVnImage;
	ipImageOut			: ITcVnImage;
	// Displayable Images
	ipImageInDisplay		: ITcVnDisplayableImage;
	ipImageOutDisplay		: ITcVnDisplayableImage;
	ipImageContourDisp		: ITcVnDisplayableImage;

	ipContours			: ITcVnContainer;
	aOffset				: TcVnPoint;
	ipImageContour			: ITcVnImage;
	aRedColor			: TcVnVector4_LREAL := [255, 0, 0, 0];
	aGreenColor			: TcVnVector4_LREAL := [0, 255, 0, 0];
	stRectangle			: TcVnRectangle_UDINT;
	fThreshold			: LREAL := 150;
	eThresholdType			: ETcVnThresholdType :=
ETcVnThresholdType.TCVN_TT_TOZERO;
	ipForwardIterator		: ITcVnForwardIterator;
	ipTempContainer			: ITcVnContainer;
	fTempArea			: LREAL;
	fBiggestArea		: LREAL;
	nBiggestIndex		: ULINT;
	nIndex				: ULINT;
	fWidhtMul			: LREAL := 0.3;
	fHeightMul			: LREAL := 0.2;
	nImageWidth			: UDINT;
	nImageHeight		: UDINT;
	nRoiMinWidth			: UDINT;
	i,j				: INT;
	stPicture			: ARRAY [0..27, 0..27] OF BYTE;
	stPictureEx			: ARRAY [0..27, 0..27] OF INT;
END_VAR

IF bReset THEN
bReset := FALSE;
MEMSET(ADR(GVL.stPicture), 255, SIZEOF(ST_Picture));
MEMSET(ADR(GVL.arrPredictions), 0 , SIZEOF(GVL.arrPredictions));
END_IF

eState := fbFileSource.GetState();

CASE eState OF
TCVN_CS_INITIAL, TCVN_CS_OPENING, TCVN_CS_OPENED,
TCVN_CS_STARTACQUISITION:
```

```
fbFileSource.StartAcquisition();

TCVN_CS_TRIGGERING:
fbFileSource.TriggerImage();

TCVN_CS_ACQUIRING:
IF bTriggerImage THEN
hr      := fbFileSource.TriggerImage();
IF SUCCEEDED(hr) THEN
bTriggerImage          := FALSE;
MEMSET(ADR(GVL.arrPredictions), 0 , SIZEOF(GVL.arrPredictions));
END_IF
ELSE
hr      := fbFileSource.GetCurrentImage(ipImageIn);
IF SUCCEEDED(hr) AND ipImageIn <> 0 THEN
// Renkli tonlardan gri tonuna gecme
hr      := F_VN_ConvertColorSpace(ipImageIn, ipImageIn,
EtcVnColorSpaceTransform.TCVN_CST_BAYER_RG_TO_GRAY, hr);
hr      := F_VN_CopyIntoDisplayableImage(ipImageIn, ipImageInDisplay, hr);

// Esik oncesi renkleri ters cevirme
hr      := F_VN_InvertImageColor(ipImageIn, ipImageOut, hr);
// Esik uygulayarak gri tonlarindan kurtulma
hr      := F_VN_Threshold(ipImageOut, ipImageOut, fThreshold, 255,
eThresholdType, hr);

// Imajin boyutlarini alma
hr := F_VN_GetImageWidth(ipImageOut, nImageWidth, hr);
hr := F_VN_GetImageHeight(ipImageOut, nImageHeight, hr);

// Ilgi bolgesi bulma
hr := F_VN_FindContoursExp(ipImageOut, ipContours,
ETcVnContourRetrievalMode.TCVN_CRM_EXTERNAL,
ETcVnContourApproximationMethod.TCVN_CAM_SIMPLE, aOffset, hr);

hr := F_VN_ConvertColorSpace(ipImageOut, ipImageContour,
ETcVnColorSpaceTransform.TCVN_CST_GRAY_TO_RGB, hr);
hr := F_VN_DrawContours(ipContours, -1, ipImageContour, aRedColor, 3, hr);

hr := F_VN_GetForwardIterator(ipContours, ipForwardIterator, hr);

nIndex := -1; fBiggestArea := -1;
WHILE SUCCEEDED(hr) AND_THEN ipForwardIterator.CheckIfEnd() <> S_OK
DO
hr := F_VN_GetContainer(ipForwardIterator, ipTempContainer, hr);
hr := F_VN_IncrementIterator(ipForwardIterator, hr);

IF SUCCEEDED(hr) THEN
nIndex := nIndex + 1;
END_IF
```

```
// Objenin alanini al
hr := F_VN_ContourArea(ipTempContainer, fTempArea, hr);
// En buyuk alanin belirlenmesi
IF fTempArea > fBiggestArea THEN
fBiggestArea := fTempArea;
nBiggestIndex := nIndex;
END_IF
END_WHILE

// Bulunan en buyuk alani al
hr := F_VN_GetAt_ITcVnContainer(ipContours, ipTempContainer, nBiggestIndex,
hr);
// Bu alani kapsayan bir dikdortgen ciz
hr := F_VN_UprightBoundingRectangle(ipTempContainer, stRectangle, hr);

IF SUCCEEDED(hr) THEN
IF stRectangle.nX - TO_UDINT(stRectangle.nWidth * fWidhtMul) < 5000 THEN
stRectangle.nX := stRectangle.nX - TO_UDINT(stRectangle.nWidth * fWidhtMul);
ELSE
stRectangle.nX := 0;
END_IF

IF stRectangle.nY - TO_UDINT(stRectangle.nHeight * fHeightMul) < 5000 THEN
stRectangle.nY := stRectangle.nY - TO_UDINT(stRectangle.nHeight * fHeightMul);
ELSE
stRectangle.nY := 0;
END_IF

IF (stRectangle.nX + stRectangle.nWidth + TO_UDINT(stRectangle.nWidth * 2 *
fWidhtMul)) > nImageWidth THEN
stRectangle.nWidth := nImageWidth - stRectangle.nX - 1;
ELSE
stRectangle.nWidth := stRectangle.nWidth + TO_UDINT(stRectangle.nWidth * 2 *
fWidhtMul);
END_IF

IF stRectangle.nWidth < nRoiMinWidth THEN
stRectangle.nWidth := nRoiMinWidth;
END_IF

IF (stRectangle.nY + stRectangle.nHeight + TO_UDINT(stRectangle.nHeight * 2 *
fHeightMul)) > nImageHeight THEN
stRectangle.nHeight := nImageHeight - stRectangle.nY - 1;
ELSE
stRectangle.nHeight := stRectangle.nHeight + TO_UDINT(stRectangle.nHeight * 2
* fHeightMul);
END_IF
END_IF
```

```
hr := F_VN_DrawRectangle_TcVnRectangle_UDINT(stRectangle, ipImageContour,
aGreenColor, 3, hr);
hr := F_VN_CopyIntoDisplayableImage(ipImageContour, ipImageContourDisp, hr);
hr := F_VN_SetRoi_TcVnRectangle_UDINT(stRectangle, ipImageOut, hr);

/////////
// Imaji 28x28 yap
hr      := F_VN_ResizeImage(ipImageOut, ipImageOut, 28, 28,
EtcVnInterpolationType.TCVN_IT_NEAREST_NEIGHBOR, hr);
// Tekrar esik degeri uygula
hr      := F_VN_Threshold(ipImageOut, ipImageOut, 180, 255,
ETcVnThresholdType.TCVN_TT_BINARY, hr);
// Tonlamayi ters cevir
hr      := F_VN_InvertImageColor(ipImageOut, ipImageOut, hr);
hr      := F_VN_CopyIntoDisplayableImage(ipImageOut, ipImageOutDisplay, hr);
// Byte dizisi seklinde PLC degiskenine ata
hr      := F_VN_ExportImage(ipImageOut, ADR(stPicture) , SIZEOF(stPicture),
hr);

// Python byte okurken sorun yasadigi icin int kullanilmistir
FOR i := 0 TO 27 BY 1 DO
FOR j := 0 TO 27 BY 1 DO
stPictureEx[i,j]        := TO_INT(stPicture[i,j]);
END_FOR
END_FOR
END_IF
END_IF

TCVN_CS_ERROR:
fbFileSource.Reset();
END_CASE
```

ÖZGEÇMİŞ

Ad-Soyad **:** Göktuğ ÜLKÜER
Doğum Tarihi ve Yeri **:** 28.09.1993, İstanbul
E-posta **:** ulkuer@itu.edu.tr

ÖĞRENİM DURUMU:

- **Lisans** **:** 2016, İstanbul Teknik Üniversitesi, Elektrik Elektronik Fakültesi, Kontrol ve Otomasyon Mühendisliği

MESLEKİ DENEYİM VE ÖDÜLLER:

- 2016 – 2020 : Beckhoff Otomasyon Ltd. Şti.