**BAŞKENT ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**
**ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI**
**ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI**

# FPGA TABANLI DERİN ÖĞRENME ALGORİTMASI İLE BASKILI DEVRE KUSURLARININ TESPİTİ

**HAZIRLAYAN**

**ENES EMRE ÖNER**

**YÜKSEK LİSANS TEZİ**

**ANKARA - 2025**

**BAŞKENT ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**
**ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI**
**ELEKTRİK ELEKTRONİK MÜHENDİSLİĞİ TEZLİ YÜKSEK LİSANS PROGRAMI**

# FPGA TABANLI DERİN ÖĞRENME ALGORİTMASI İLE BASKILI DEVRE KUSURLARININ TESPİTİ

**HAZIRLAYAN**

**ENES EMRE ÖNER**

**YÜKSEK LİSANS TEZİ**

**TEZ DANIŞMANI**

**DOÇ. DR. SELDA GÜNEY**

**ANKARA - 2025**

**BAŞKENT ÜNİVERSİTESİ**
**FEN BİLİMLERİ ENSTİTÜSÜ**

Elektrik-Elektronik Mühendisliği Anabilim Dalı Elektrik-Elektronik Mühendisliği Tezli Yüksek Lisans Programı çerçevesinde Enes Emre Öner tarafından hazırlanan bu çalışma, aşağıdaki jüri tarafından Yüksek Lisans Tezi olarak kabul edilmiştir.

Tez Savunma Tarihi: 07/07/2025

**Tez Adı:** FPGA Tabanlı Derin Öğrenme Algoritması ile Baskılı Devre Kusurlarının Tespiti

| **Tez Jüri Üyeleri (Unvanı, Adı- Soyadı, Kurumu)** | | **İmza** |
|---|---|---|
| Prof. Dr. Hamit Erdem | Başkent Üniversitesi | …… |
| Doç. Dr. Derya Yılmaz | Gazi Üniversitesi | …… |
| Doç. Dr. Selda Güney | Başkent Üniversitesi | …… |

**ONAY**

Prof. Dr. Dilek Çökeliler Serdaroğlu

Fen Bilimleri Enstitüsü Müdürü

…….

Tarih: … / … / 2025

**BAŞKENT ÜNİVERSİTESİ**

**FEN BİLİMLER ENSTİTÜSÜ**

**YÜKSEK LİSANS TEZ ÇALIŞMASI ORİJİNALLİK RAPORU**

Tarih: … / … / 20…

Öğrencinin Adı, Soyadı : Enes Emre Öner

Öğrencinin Numarası : 22210423

Anabilim Dalı : Elektrik-Elektronik Mühendisliği

Programı : Tezli Yüksek Lisans

Danışmanın Unvanı/Adı, Soyadı : Doç. Dr. Selda Güney

Tez Başlığı : FPGA Tabanlı Derin Öğrenme Algoritması ile Baskılı Devre Kusurlarının Tespiti

Yukarıda başlığı belirtilen Yüksek Lisans tez çalışmamın; Giriş, Ana Bölümler ve Sonuç Bölümünden oluşan, toplam 99 sayfalık kısmına ilişkin, 29/07/2025 tarihinde tez danışmanım tarafından Turnitin adlı intihal tespit programından aşağıda belirtilen filtrelemeler uygulanarak alınmış olan orijinallik raporuna göre, tezimin benzerlik oranı %7'dir. Uygulanan filtrelemeler:

1. Kaynakça hariç
2. Alıntılar hariç
3. Beş (5) kelimeden daha az örtüşme içeren metin kısımları hariç

"Başkent Üniversitesi Enstitüleri Tez Çalışması Orijinallik Raporu Alınması ve Kullanılması Usul ve Esaslarını" inceledim ve bu uygulama esaslarında belirtilen azami benzerlik oranlarına tez çalışmamın herhangi bir intihal içermediğini; aksinin tespit edileceği muhtemel durumda doğabilecek her türlü hukuki sorumluluğu kabul ettiğimi ve yukarıda vermiş olduğum bilgilerin doğru olduğunu beyan ederim.

Öğrenci İmzası: ……

**ONAY**

Tarih: … /… / 20…

Öğrenci Danışmanı:

Doç. Dr. Selda Güney

…..……

# TEŞEKKÜR

Yüksek lisans yolculuğumun ve bu kapsamlı tez çalışmasının ışığında, her adımda sağlam bir rehber ve destek olan kıymetli tez danışmanım Doç. Dr. Selda Güney'e en içten teşekkürlerimi sunarım. Yol gösterici bilgelikleriniz, sabrınız ve samimiyetiniz, bu zorlu süreçte benim için bir ışık kaynağı oldu. Ek olarak, bana yüksek lisans yolculuğum boyunca sağladıkları destekten dolayı annem Nilüfer Öner, babam Kadir Ramadan Öner ve kardeşim Tuana Öner'e en samimi şekilde teşekkürlerimi sunarım. Son olarak, bu yolculuğumda her zaman yanımda olduklarını hissettiren ve kendi tecrübelerini benimle paylaşarak bana destek olan Eyüphan İpek'e ve Derya Çetin'e en yürekten teşekkürlerimi sunarım.

# ÖZET

**Enes Emre ÖNER**

**FPGA TABANLI DERİN ÖĞRENME ALGORİTMASI İLE BASKILI DEVRE KUSURLARININ TESPİTİ**

**Başkent Üniversitesi Fen Bilimleri Enstitüsü**

**Elektrik-Elektronik Mühendisliği Anabilim Dalı**

**2025**

Endüstrinin gelişmesiyle birlikte hayatımızın her noktasında bulunan elektronik donanımların temel altyapısı olan elektronik kartların ve devre yollarının küçülmesi, baskılı devre üretimini karmaşıklaştırarak, son üründe kusurların meydana gelmesine sebep olabilmektedir. Baskılı devrede oluşabilecek kusurlar elektronik kartların küçük olması sebebiyle dikkatlice, büyüteç gibi ekipmanlar kullanılarak kontrol edilmelidir. Bu yöntem, zamansal olarak uzundur ve hatalara açıktır. Kusurların tespit edilememesi cihazın kısa sürede arıza vermesine ve performansının düşük seviyelerde seyretmesine sebep olarak kullanıcıyı mağdur edecek ve ürün/marka taleplerinde düşüşe sebep olacaktır. Bu soruna çözüm olarak, nesne tespit alanında kullanılan derin öğrenme modelleri ile çalışan sistemlerin üretim veya üretim sonrası süreçlere dahil edilmesi önerilebilir. Bu sistemler, baskılı devreler üzerinde oluşan kusurları, yüksek doğrulukta ve yüksek hızda tespit edilebilecek kabiliyete sahip olmalıdır. FPGA'lar (Field Programmable Gate Array), paralel işlem yeteneği, tasarımda esneklik ve yeniden programlanabilme kabiliyetleri sayesinde bu tip gerçek zamanlı çalışması gereken sistemler için genelde kullanılan CPU (Central Processing Unit), GPU (Graphical Processing Unit) ve hibrit yapılara (SoC- System on Chip) göre daha uygun bir ortam sağlamaktadır. Bu tez çalışmasında, baskılı devre üretimi sonrasında oluşan kusurların derin öğrenme modeli yardımı ile tespitini gerçekleştiren FPGA tabanlı yöntem önerilmektedir. Çalışmada kullanılan açık kaynaklı veri seti, 9666 adet görüntü ve 7 farklı kusura ait sınıf içermektedir. Yapılan çalışmada, derin öğrenme modeli olarak bu alanda FPGA ile kullanılan en son versiyon YOLOv8 modelinin daha üst versiyonlarından birisi olan YOLOv11 versiyonu kullanılmıştır. Performans farklarının anlaşılması adına YOLOv8 ve YOLOv11 modelleri aynı koşullar altında eğitilmiş ve performansları karşılaştırılmıştır. Parametre-performans ilişkisinde nihai eğitimde %98,4 mAP50 (IoU - Intersection over Union - eşiğinde ortalama hassasiyet 0,50) performansıyla

üstün gelerek gömülü sistemlerde daha az yer kaplayan YOLOv11n modelinin diğer çalışmalardan farklı olarak GPU veya CPU desteği olmaksızın sadece RTL (Register Transfer Language) tabanlı FPGA mimarisi oluşturularak sonuçların performansı incelenmiştir. İncelemeler doğrultusunda genel mAP50 değerinin %97,86 seviyesinde korunduğu, çıkarım hızının ise 34,9386306ms olduğu gözlemlenmiştir. Tespit yeteneğinin çok az miktarda kayıpla korunabilmesi, YOLOv11n modelinin RTL temelli gerçeklemesinin mümkün olduğunu göstermiştir. Bu tez çalışmasında alınan sonuçlar, gerçek zamanlı sistemlerde CPU, GPU ve hibrit yapılar yerine işlem gücü olarak sadece FPGA bulunduran sistemler kullanılabileceğine dair ışık tutmuştur.

**ANAHTAR KELİMELER:** Derin Öğrenme, FPGA, Baskılı Devre Kusurları, Kusur Tespiti, YOLO

# ABSTRACT

**Enes Emre ÖNER**

**FPGA BASED PRINTED CIRCUIT BOARD DEFECT DETECTION WITH A DEEP LEARNING ALGORITHM**

**Başkent University Institute of Science and Engineering**

**Department of Electrical and Electronics Engineering**

**2025**

With the developing industry, the miniaturization of electronic boards and circuit paths, which are the basis of electronics in our lives, complicates PCB (Printed Circuit Board) production and can cause defects on the final product. Defects occurring on the PCB should be checked carefully using equipment such as a magnifying glass due to the size of the electronic cards. This method is time-consuming and prone to mistakes. Failure of detection of defects may cause the device to fail in a short time and its performance to remain at low levels, which will cause a decrease in product/brand demand by victimizing the user. As a solution, including systems working with deep learning models, used in object detection, in production or post-production stages can be suggested. These systems must detect defects on PCBs with high accuracy and speed. FPGAs (Field Programmable Gate Array) provide a more suitable environment for real-time systems, with their parallel processing capability, flexibility in design and re-programmability, compared to CPU (Central Processing Unit), GPU (Graphical Processing Unit) and hybrid structures (SoC - System on Chip) which are generally used in these systems. In this thesis, an FPGA-based architecture is proposed that detects defects, which occur after PCB production, with the help of a deep learning model. The open-source dataset used in this study contains 9666 images and 7 different defect classes. In the thesis, one of the upper versions of the YOLOv8 model, the YOLOv11 version, is used as the deep learning model, which is the latest version used in this with FPGA field so far. To understand the performance differences, YOLOv8 and YOLOv11 models are trained under the same conditions and their performances are compared. In the parameter-performance relationship, the YOLOv11n model, which occupies less space in embedded systems and achieving 98,4% mAP50 (mean average precision for IoU - Intersection over Union – at threshold 0,50) performance in the final training, is examined by creating only RTL (Register Transfer Language) based FPGA architecture without GPU

or CPU support. Within the observations in examinations, the overall mAP50 value is preserved at 97,86% with the inference speed is 34,9386306ms. The fact of the preservation of detection capability with a very small amount of loss proves the RTL implementation of YOLOv11n model is possible. The results in this thesis shed light on the systems contains FPGA as processing power can be used instead of CPU, GPU and hybrid structures in real-time systems.

**KEYWORDS:** Deep Learning, FPGA, PCB Defects, Defect Detection, YOLO

# ÖNSÖZ

Gelişen teknoloji ile birlikte elektronik kartların boyutlarının küçültülerek daha karmaşık devre tasarımlarının oluşması sağlanabilmiştir. Bu kartların üretimi veya üretimi sonrasında oluşabilecek kusurlar da dolayısıyla daha mümkün mertebeye gelmiştir. Bu sebeple, bu kusurların tespit edilerek kusurlu kartların piyasaya sürülmemesi üretici açısından büyük önem arz etmektedir. Bu tez çalışması, üretilmiş ve dizgisi yapılmamız baskı devre kartlarının üzerindeki kusurların hızlı ve yüksek doğrulukta tespit edilebilmesi adına derin öğrenme modeli kullanılarak yürütülmüştür. Ek olarak, derin öğrenme modelini hızlandırabilmek adına FPGA tabanlı bir tasarım gerçekleştirilmiştir. Derin öğrenme modeli ile elde edilen yüksek eğitim ve test başarıları, kullanılan veri setleri üzerinde etkileyici bir performans sergilemektedir. Yüksek lisans eğitimim süresince bilgi birikimini benimle paylaşmaktan çekinmeyen, çalışmalarıma yol gösteren, güvenen ve moral ve motivasyon yardımlarını eksik etmeyen çok değerli hocam Doç. Dr. Selda Güney'e teşekkürlerimi sunarım.

# İÇİNDEKİLER

**TEŞEKKÜR** ..... i
**ÖZET** ..... ii
**ABSTRACT** ..... iv
**ÖNSÖZ** ..... vi
**İÇİNDEKİLER** ..... vii
**TABLOLAR LİSTESİ** ..... ix
**ŞEKİLLER LİSTESİ** ..... x
**SİMGELER VE KISALTMALAR** ..... xiii
**1. GİRİŞ** ..... 1
**1.1. Çalışmanın Amacı** ..... 3
**1.2. Tezin Organizasyonu** ..... 3
**2. LİTERATÜRDEKİ ÇALIŞMALAR** ..... 5
**3. KULLANILAN VERİ SETİ VE MAKİNE ÖĞRENMESİ** ..... 16
**3.1. Veri Setleri** ..... 17
**3.1.1. PCB-DATASET** ..... 18
**3.1.2. DeepPCB** ..... 18
**3.1.3. TDD-PCB** ..... 19
**3.1.4. PCBYOLO8** ..... 20
**3.1.5. PCBA-DET** ..... 21
**3.1.6. detection-pcb-defects-yolov8** ..... 22
**3.2. Yapay Sinir Ağları** ..... 24
**3.3. Derin Öğrenme** ..... 25
**3.4. Evrişimsel Sinir Ağları (CNN)** ..... 25
**3.5. CNN Algoritmalarının Katmanları** ..... 27
**3.5.1. Giriş Katmanı (Input layer)** ..... 27
**3.5.2. Evrişim Katmanı (Convolutional layer)** ..... 27
**3.5.3. Aktivasyon Fonksiyonu (Activation function)** ..... 28
**3.5.4. Havuzlama Katmanı (Pooling layer)** ..... 32
**3.5.5. Tam Bağlı Katman (Fully connected layer)** ..... 32
**3.5.6. Seyreltme Katmanı (Dropout layer)** ..... 33
**3.5.7. Sınıflandırma Katmanı (Classification layer)** ..... 34
**3.6. You Only Look Once Version 11 (YOLOv11) Algoritma Katmanları** ..... 34

3.6.1. Omurga (Backbone) ..... 38
3.6.1.1. Evrişimsel Katmanlar ..... 39
3.6.1.2. C3k2 Bloğu ..... 40
3.6.2. Boyun (Neck) ..... 42
3.6.2.1. SPPF ve C2PSA ..... 43
3.6.2.2. Dikkat Mekanizması (Attention Mechanizm) ..... 46
3.6.3. Kafa (Head) ..... 47
3.7. Performans Değerlendirme Ölçütleri ..... 48
4. ALANSAL PROGRAMLANABİLİR KAPI DİZİSİ (FPGA) ..... 53
4.1. RAM Yapısı ..... 55
4.2. FIFO Yapısı ..... 57
4.3. Kayan Noktalı Gösterim IP Çekirdeği ..... 59
5. UYGULANAN YÖNTEMLER ..... 65
5.1. YOLOv8 ve YOLOv11 Eğitimi ve Karşılaştırılması ..... 67
5.2. YOLOv11 Hiper Parametrelerinin Belirlenmesi ..... 69
5.2.1. Devir (Epoch) Sayısına Göre Performans Kıyaslaması ..... 70
5.2.2. Öğrenme Oranına (Learning Rate) Göre Performans Kıyaslaması ..... 74
5.2.3. L2 Düzenlileştirme Oranına (L2 Regularization Rate - Weight Decay) Göre Performans Kıyaslaması ..... 75
5.3. YOLOv11 Modelinin FPGA'ya Gerçeklenmesi ve Sonuçların Alınması ..... 75
5.3.1. Evrişim Ağırlıkları ile Batch Normalizasyonu Ağırlıklarının Katlanması ..... 77
5.3.2. Ağırlıkların 16-bit Kayan Noktalı Gösterim Formatına Dönüştürülmesi ..... 77
5.3.3. Simülasyon Ortamının Hazırlanması ..... 78
5.3.4. Modelin FPGA'da Gerçeklenmesi ve Simülasyonu ..... 80
5.4. Sonuçların Karşılaştırılması ..... 95
6. SONUÇ VE ÖNERİLER ..... 98
KAYNAKLAR ..... 100

# TABLOLAR LİSTESİ

**Sayfa**

Tablo 2.1. Literatür Çalışmaları ..... 12

Tablo 3.1. Veri Setleri Özet Tablosu ..... 23

Tablo 3.2. YOLO Modellerinin Evrimi ..... 35

Tablo 3.3. Karmaşıklık Matrisi Değerlendirmesi ..... 48

Tablo 4.1. XC7K70T Serisi FPGA'da Kayan Noktalı Gösterim IP Çekirdeği Ayarlarına Göre Kaynak Tüketimi ..... 64

Tablo 5.1. Farklı YOLO Modellerinin Eğitim Kıyaslaması ..... 68

Tablo 5.2. Farklı Optimizasyon Yöntemlerinin Farklı Devirlerdeki Performansı ..... 71

Tablo 5.3. Farklı Öğrenme Oranlarındaki Eğitim Performansı ..... 75

Tablo 5.4. Farklı L2 Düzenlileştirme Oranlarındaki Eğitim Performansı ..... 75

Tablo 5.5. Sentez Sonrası FPGA Kaynak Tüketimi ..... 91

Tablo 5.6. FPGA ve Python Çıktıları Kıyaslaması ..... 96

Tablo 5.7. Literatürde Alınan Sonuçlar ile Performans Kıyaslaması ..... 97

# ŞEKİLLER LİSTESİ

**Sayfa**

Şekil 3.1. Makine Öğrenmesi, YSA ve Derin Öğrenme Arasındaki İlişki ..... 17

Şekil 3.2. PCB-DATASET Veri seti Görüntü Örnekleri; a) Eksik Delik, b) Fare Isırığı, c) Açık Devre, d) Kısa Devre, e) Mahmuz, f) Sahte Bakır ..... 18

Şekil 3.3. DeepPCB Veri seti Görüntü Örnekleri; a) Kısa Devre, Açık Devre, Mahmuz ve Fare Isırığı ve b) Kısa Devre, Sahte Bakır, Pin Deliği, Mahmuz ve Fare Isırığı ..... 19

Şekil 3.4. PCB-DATASET Veri seti Görüntü Örnekleri; a) Eksik Delik, b) Fare Isırığı, c) Açık Devre, d) Kısa Devre, e) Mahmuz, f) Sahte Bakır ..... 20

Şekil 3.5. PCBYOLO8 veri seti Görüntü Örnekleri; a) Eksik Delik, b) Fare Isırığı, c) Açık Devre, d) Kısa Devre, e) Mahmuz, f) Sahte Bakır, g) Çizik ..... 21

Şekil 3.6. PCBA-DET Veri Seti Görüntü Örnekleri; a) Gevşek Fan Vidaları, b) Eksik Fan Vidası, c) Gevşek Anakart Vidaları, d) Eksik Anakart Vidalar, e) Gevşek Fan Kablosu, f) Eksik Fan Kablosu, g) Fan Çizikleri, h) Anakart Çizikleri ..... 22

Şekil 3.7. detection-pcb-defects-yolov8 Veri Seti Görüntü Örnekleri; a) Eksik Delik ve Kısa Devre, b) Fare Isırığı ve Mahmuz, c) Açık Devre ve Sahte Bakır, d) Çizik ..... 23

Şekil 3.8. YSA Model Örneği ..... 24

Şekil 3.9. Genel CNN Mimarisi ..... 26

Şekil 3.10. 5x5x3 Boyutundaki Görüntüye 3x3'lük Filtre ile Konvolüsyon Örneği ..... 28

Şekil 3.11. Aktivasyon Fonksiyonu "f""in Nörondaki Yeri ve Çıkışa Etkisi ..... 29

Şekil 3.12. ReLU Fonksiyonu ve Türevi Grafiği ..... 30

Şekil 3.13. SiLU ve ReLU Fonksiyonları ..... 31

Şekil 3.14. SiLU'nun Türevi ve Sigmoid Fonksiyonu ..... 31

Şekil 3.15. Standart bir CNN ağına seyreltme işleminin uygulanması. a) Standart CNN ağ yapısı. b) Seyreltme katmanından sonraki ağ yapısı. ..... 33

Şekil 3.16. YOLOv11 Derin Öğrenme Modeli ..... 38

Şekil 3.17. YOLOv11 Omurga (Backbone) Yapısı ..... 39

Şekil 3.18. Evrişim Katmanı ..... 40

Şekil 3.19. C3k2 Bloğu ..... 41

Şekil 3.20. C3k ve Bottleneck Blokları ..... 41

Şekil 3.21. YOLOv11 Boyun (Neck) Yapısı ..... 43

Şekil 3.22. SPPF Bloğu ..... 45

Şekil 3.23. C2PSA ve PSA Blokları ..... 45

Şekil 3.24. Dikkat Mekanizması Modeli ..... 46

Şekil 3.25. YOLOv11 Kafa (Head) Yapısı ve Tespit (Detect) Bloğu ..... 48

Şekil 4.1. CLB, Giriş/Çıkış Blokları ve Ara Bağlantılar ..... 54

Şekil 4.2. Çift bağlantı noktalı Blok RAM şeması ..... 56

Şekil 4.3. AMD Xilinx Vivado Çift Bağlantılı Blok RAM IP Kataloğu ..... 57

Şekil 4.4. 36x2 Çift Bağlantılı Blok RAM İlkel Bloğu Kod Parçası ..... 57

Şekil 4.5. FIFO İç Yapısı ..... 58

Şekil 4.6. AMD Xilinx Vivado FIFO IP Kataloğu ..... 59

Şekil 4.7. Kayan Noktalı Gösterim IP Çekirdeği IP Katalog, Operasyon Seçme ..... 60

Şekil 4.8. Kayan Noktalı Gösterimde Bit Alanları ..... 61

Şekil 4.9. Kayan Noktalı Gösterim IP Çekirdeği IP Katalog - Girdinin Bit Genişliğini Ayarlama Sekmesi ..... 61

Şekil 4.10. Kayan Noktalı Gösterim IP Çekirdeği IP Katalog - Kaynak Tüketimi Sekmesi ..... 63

Şekil 4.11. Kayan Noktalı Gösterim IP Çekirdeği IP Katalog - Gecikme Ayarları Sekmesi ..... 64

Şekil 5.1. Çalışmanın Adımları ..... 66

Şekil 5.2. YOLOv8n En İyi Eğitim Sonuç Grafikleri ..... 68

Şekil 5.3. YOLOv8s En İyi Eğitim Sonuç Grafikleri ..... 68

Şekil 5.4. YOLOv11n En İyi Eğitim Sonuç Grafikleri ..... 69

Şekil 5.5. Adam Optimizasyon Yöntemi Eğitim Sonuçları ..... 71

Şekil 5.6. AdamW Optimizasyon Yöntemi Eğitim Sonuçları ..... 72

Şekil 5.7. NAdam Optimizasyon Yöntemi Eğitim Sonuçları ..... 72
Şekil 5.8. RAdam Optimizasyon Yöntemi Eğitim Sonuçları ..... 73
Şekil 5.9. SGD Optimizasyon Yöntemi Eğitim Sonuçları ..... 73
Şekil 5.10. RMSProp Optimizasyon Yöntemi Eğitim Sonuçları ..... 74
Şekil 5.11. Video Jeneratörü Simülasyonu ..... 78
Şekil 5.12. Video Jeneratörü Sonuç Görsel ..... 79
Şekil 5.13. Simülasyon Ortamı Mimarisi ..... 80
Şekil 5.14. Simülasyon Ortamı Hiyerarşisi ..... 80
Şekil 5.15. Piksel Dönüşüm Modülü Mimarisi ..... 82
Şekil 5.16. 2 Adımlı 1 Dolgulu 3x3 Evrişim Operasyonu Mimarisi ..... 83
Şekil 5.17. 1 Adımlı 1 Dolgulu 3x3 Evrişim Operasyonu Mimarisi ..... 85
Şekil 5.18. 1 Adımlı 1x1 Evrişim Operasyonu Mimarisi ..... 86
Şekil 5.19. 5x5 Maksimum Havuzlama Mimarisi ..... 86
Şekil 5.20. Üst Örnekleme (Up Sample) Mimarisi ..... 87
Şekil 5.21. DFL ile Koordinatların Belirlenme Mimarisi ..... 88
Şekil 5.22. Kutucuk Koordinatlarını Ölçeklendirme ..... 89
Şekil 5.23. Sınıf Skorlarını Belirleme Mimarisi ..... 90
Şekil 5.24. Çatı Modülü Mimarisi ..... 90
Şekil 5.25. Çatı Modül Hiyerarşisi ..... 91
Şekil 5.26. Simülasyon Üzerinde Tespit Süresi ..... 92
Şekil 5.27. Sınırlayıcı Kutu Değerleri Çıktıları ..... 93
Şekil 5.28. a) Sınıf Çıktıları Başlangıcı ve Değerleri b) Bütün Sınıf Çıktıları Süreci ..... 94
Şekil 5.29. FPGA Tahmin Çıktısının Görselleştirilmiş Hali; a) pinhole, b) opencircuit ve false copper, c) mousebite ve short circuit, d) pinhole ve scratch ..... 95

# SİMGELER VE KISALTMALAR

| | |
|---|---|
| AOI | Automatic Optical Inspection |
| AP | Average Precision |
| AVI | Automatic Visual Inspection |
| C2PSA | Cross Stage Partial with Spatial Attention |
| CLB | Configurable Logic Blocks |
| CNN | Convolutional Neural Network |
| CPU | Central Processing Unit |
| CUDA | Compute Unified Device Arcthitectur |
| CV | Computer Vision |
| DFL | Distribution Focal Loss |
| DN | Doğru Negatif |
| DP | Doğru Pozitif |
| DSP | Digital Signal Proccessor |
| DWConv | Deptwise Convolution |
| FIFO | First-in-First-out |
| FN | False Negative |
| FP | False Positive |
| FPGA | Field Programmable Gate Array |
| FPN | Feature Pyramid Network |
| FPS | Frame Per Second |
| FRCNN | Faster Recursive Convolutional Neural Network |
| GAN | Generative Adversarial Networks |
| GELAN | Generalized Efficent Layer Aggregation Networks |
| GPP | Group Pyramid Pooling |
| GPU | Graphical Processing Unit |
| HDL | Hardware Description Language |
| HLS | High Level Synthesis |
| HOG | Histogram of Oriented Gradients |
| HSV | Hue Saturation Value |
| I/O | Input/Output |
| IDE | Integrated Development Environment |
| IEEE | Institute of Engineers and Everyone Else |
| IOB | Input Output Block |
| IoU | Intersection over Union |
| IP | Intellectual Property |
| KNN | K-Nearest Neighbors |
| LUT | Look-Up Table |
| mAP | Mean Average Precision |
| MATLAB | Matrix Laboratory |
| MB | Mega Byte |
| MHz | Mega Hertz |
| ML | Machine Learning |
| NMS | Non Max Suppression |
| PANet | Path Aggregation Network |
| PCB | Printed Circuit Board |
| PGI | Programmable Gradient Information |
| PLD | Programmable Logic Devices |

| | |
|---|---|
| PSA | Partial Spatial Attention |
| QSPI | Quad Serial Peripheral Interface |
| RAM | Random Access Memory |
| RCNN | Recursive Convolutional Neural Networks |
| ReLU | Rectified Linear Unit |
| RGB | Red Green Blue |
| ROM | Read Only Memory |
| RPN | Region Proposal Network |
| RTL | Register Transfer Language |
| SGD | Stokastik Gradient Descent |
| SiLU | Sigmoid Weighted Linear Unit |
| SoC | System-on-Chip |
| SPI | Serial Peripheral Interface |
| SPPF | Spatial Pyramid Pooling Fast |
| SSD | Single Shot Multibox Detector |
| SVM | Support Vector Machine |
| TN | True Negative |
| TP | True Positive |
| VHDL | Very High Speed Hardware Description Language |
| WKDE | Weighted Kernel Density Estimation |
| YN | Yanluş Negatif |
| YOLO | You Only Look Once |
| YP | Yanlış Pozitif |
| YSA | Yapay Sinir Ağları |

# 1. GİRİŞ

Elektronik kartlar hayatımızda yoğunca kullandığımız elektronik cihazların temel malzemesidir. Sıklıkla kullanılan, en alt seviye elektronik cihazların dahi sahip olduğu baskılı devre kalitesi ve üretim verimliliği cihazın performansını ve maliyetini direkt olarak etkilemektedir. Baskılı devre, fiberglas, kompozit epoksi ve lamine malzemeden üretilen, aynı zamanda transistörler, çipler, kapasitörler, indüktörler gibi elektronik bileşenlerin birlikte bulunabilmesi ve bir devrenin parçası olabilmelerini için bir temel oluşturur [1,2,3,4]. Baskılı devre üretimi esnasında çok ufak da olsa baskılı devreye zarar vermek veya üretim esnasında yaşanan bir aksaklık olma durumu çok olağandır. Bu yüzden üretim sırasında yaşanan yanlış idame ve teknik problemlerden dolayı çoğu baskılı devre kusurlu olarak üretilir. Elektronik devre tasarımında çizilen devre yollarının ve kart boyutlarının iyice küçülmesi dolayısıyla üretim sonrası oluşan kusurların göz ile tespiti oldukça zorlaşmıştır. Fakat bu kusurların tespit edilmesi ve kusurlara sahip olan baskılı devrelerin ayrılması gerekir. Eski zamanlarda bu kusurların tespiti için kullanılan geleneksel yöntemler temel olarak manuel tespit yöntemleridir [5]. Bu yöntemler günümüz teknolojisine kıyasla daha az etkili ve yavaşlardır [6].

Günümüzde, baskılı devre üzerinde oluşan kusurları tespit etmek adına birçok çalışma gerçekleştirilmiştir ve gerçekleştirilmeye devam etmektedir. Gelişen teknoloji ile birlikte baskılı devre imalat endüstrileri birbirinden farklı görüntü işleme yöntemlerini kullansa da kalite kontrol sürecine hala ayak uyduramamaktadırlar [7]. Literatürde baskılı devre kusurlarının tespitinde farklı metotlar kullanılmaktadır [8,9,10]. Özellikle, arka planda çalışan büyük bir algoritma olmamasından dolayı şablon-eşleme metodu kusur tespiti için kullanılan yöntemlerdendir [11]. Bir diğer metot ise görüntü çıkarma metodudur [12]. Aynı zamanda lehimli alanların tespiti için de kullanılan renk uzayı değişimine bağlı tespit yöntemleri de bulunmaktadır [13]. Çapraz korelasyon yöntemi de bir diğer kusur tespiti metotlarındandır [14]. Ancak bu metotlar baskılı devrelerdeki belirli kusur türlerini tespit etmeye yöneliktir. Evrişimsel sinir ağlarının (CNN) kullanılması ile görüntü tanıma ve obje tespiti açısından gözle görülür bir gelişme yaşanmıştır [10]. Bu sayede kusur tespit çeşitliliği sayısı oldukça yükselmiştir.

Bu yöntemlerin kullanılması için pahalı AVI (Automatic Visual Inspection) ve AOI (Automatic Visual Inspection) makineleri üretilmiştir. Baskılı devre üreticileri geleneksel denetim sürecinin ardından kalite denetimi için büyük miktarda insan gücünü eğitmek ve

dahil etmek zorunda kalmışlardır [15, 16]. Baskılı devrelerdeki kusurların eksiksiz tespiti için gerçekten tecrübeli ve gözü keskin çalışanlar gerekmektedir. Tabii ki bu da yıllar süren tecrübe ile sabittir. Öyle olsa bile üretim sonrası inceleme süreci yavaş geçmektedir ve insan hatası her zaman mevcuttur. Bu durumun üstesinden gelebilmek adına hatayı minimize etmek ve tespit hızını artırmak adına en popüler nesne tespit algoritmalarından birisi olan YOLO (You Only Look Once)'nun en son versiyonu olan YOLOv11 modeli bu çalışmada kullanılmıştır.

Makine öğrenmesi metotları çoğu zaman bir insana göre çok daha hızlı ve doğru sonuçlar vermektedir. Diğer makine öğrenmesi algoritmalarının yanı sıra YOLO, çerçeve tespitini bir regresyon problemi olarak gördüğünden dolayı oldukça hızlıdır. Eğitim ve test esnasında görüntüyü bir bütün olarak gördüğü için sınıfların görünümleri ile ilgili bağlamsal bilgileri örtülü olarak kodlar ve nesnelerin genelleştirilebilir temsillerini öğrenir [17]. Daha önceki çalışmalarda YOLO algoritmasının farklı versiyonları kullanılarak hatalı baskılı devre veri setleri eğitilip çok iyi sonuçlar elde edilebilmiştir [5,18,19,20,21]. Kullanılan farklı YOLO versiyonlarından farklı olarak YOLOv11, gelişmiş özellik çıkarma tekniklerini entegre ederek, daha az parametre kullanımıyla daha ince ayrıntıları yakalamayı mümkün kılar. Bu yaklaşım, nesne algılama ve sınıflandırma gibi çeşitli bilgisayarlı görme (CV) görevlerinde doğruluğu artırır. Ayrıca YOLOv11, işlem hızında kayda değer iyileştirmeler sağlayarak gerçek zamanlı performans yeteneklerini önemli ölçüde geliştirmiştir [22].

YOLOv11, CPU (Central Processing Unit), GPU (Graphical Processing Unit) ve FPGA (Field Programmable Gate Array)'lar içerisinde çalıştırılabilir. FPGA, paralel işlem yapabilme özellikleri ve tekrar tekrar programlanabilme, yüksek hızda işlem yapabilme kabiliyetleri dolayısıyla günümüzde olukça yoğun bir şekilde tercih edilmektedir [23]. FPGA'lar yeniden programlanabilir, görüntü üzerinde operasyon yapmaya uygun çok sayıda mantıksal hücre uygulamasına sahiptir. Görüntüyü segmente eden YOLO katmanlarını hızlandırmak amacıyla FPGA'nın paralelliği ve boru hattı işlevselliği kullanılabilir. Bu özellikleri sayesinde mikro işlemci gruplarına nazaran aynı anda daha fazla paralel işlem yapılarak hesaplama içeren algoritmalara hız kazandırabilirler. Aynı zamanda gerçek zamanlı işlem açısından kameradan gelen görüntü neredeyse anlık olarak analiz edilebilir [24]. Ancak, FPGA'in avantajlarının yanı sıra geliştirme karmaşıklığı CPU ve GPU sistemlerine nispeten daha karmaşıktır, halka açık veri elde edilmesi bakımından daha zor bir cihaz türüdür ve YOLO gibi büyük modeller için yeterli depolama kaynağı bulunmadığından dolayı ek hafıza birimi ihtiyacı bulunmaktadır. Bütün bu avantaj ve

dezavantajlar göz önünde bulundurularak YOLO gibi derin öğrenme modellerinin gerçeklenmesi gereken cihaza karar verilmesi gerekmektedir.

Bu tez çalışmasında FPGA tabanlı olarak önceden eğitilmiş YOLOv11 algoritma modeli çalıştırılarak baskılı devre hatalarının tespiti gerçekleştirilmiştir. Bu çalışmanın literatüre katkısı şu şekildedir, ilk olarak literatürdeki çeşitli kusurlu baskılı devreleri içeren veri setleri incelenip içlerinden çeşitli ve sağlıklı verilere sahip olan veri seti sunulmuştur. İkinci olarak, FPGA içerisinde sadece RTL (Register Transfer Language) tabanlı tasarım yapılarak YOLOv11 modelinin entegre edilişine dair bir tasarım önerisi sunulmuştur. Son olarak, FPGA tabanlı YOLOv11 modeli test edilerek, kaynak tüketimi, modelin tespit süresi, maksimum saniyedeki çerçeve sayısı ve doğruluk yüzdesi bilgileri sunulmuştur.

## 1.1. Çalışmanın Amacı

Tezin amacı, günümüzde popüler olan nesne tespit algoritmalarından birisi olan YOLO algoritmasının on birinci versiyonu YOLOv11 modeli detection-pcb-defects-yolov8 adlı veri seti ile eğitilerek eğitilmiş modelin FPGA tabanlı çalıştırılması ile üretilmiş olan baskılı devrelerdeki kusurları algılayabilen, sınıflandırabilen ve gerçek zamanlı uygulamalarda başarılı performans sergileyebilen gömülü bir sistem tasarlamaktır. Elde edilecek sonuçlar ile birlikte, üretilmiş baskılı devrelerdeki üretim veya sonradan oluşabilecek kusurları tespit etme konusunda YOLOv11'in FPGA ve sadece RTL tabanlı olarak çalıştırılmasının etkisi ve YOLOv11'in genel başarısı katkısı sağlanacaktır.

## 1.2. Tezin Organizasyonu

Açık kaynaklı bir veri setinin derin öğrenme yöntemlerinden birisi olan YOLOv11 kullanılarak eğitilmesi sonucu eğitilmiş modelin FPGA tabanlı olarak çalıştırılarak baskılı devre hatalarındaki kusurların tespitini amaçlayan bu tez çalışması, aşağıdaki gibi altı farklı bölümden oluşacak şekilde düzenlenmiştir.

Çalışmanın ilk bölümünde tez çalışmasının amacı, motivasyonu ve baskılı devre kusurlarının tespiti ile ilgili geçmişte uygulanan yöntemler ve günümüzdeki yöntemler hakkında bilgiler verilmiş, tezin organizasyonu sunulmuştur.

Çalışmanın ikinci bölümünde baskılı devredeki kusurların tespitine yönelik farklı veri setleri veya yöntemler kullanılan çalışmalara, bu çalışmaların yöntemlerinin içeriklerine, çalışmaların gerçekleştirildiği cihazlara ve çalışmaların performans sonuçlarına değinilmiştir.

Çalışmanın üçüncü bölümünde tez çalışmasında kullanılma ihtimali olan veri setlerinin içerikleri açıklanmış, ayrıca veri setleri arasından tezde kullanılan veri setinin seçilme sebepleri sunulmuştur. Bütün aday veri setleri detaylıca anlatılmış ve veri setleri içerisinden örnek görüntüler sunulmuştur. Ayrıca yapay sinir ağları ile birlikte bu tez çalışması süresince kullanılan derin öğrenme sisteminin alt sistemleri anlatılmış ve çalışma prensipleri detaylandırılmıştır.

Çalışmanın dördüncü bölümünde Alansal Programlanabilir Kapı Dizileri veya bukalemun yongaları (FPGA) teknolojisinden bahsedilmiş, iç mimarisi genel olarak açıklanmış ve FPGA içerisinde bulunan hafıza birimlerinden RAM ve FIFO yapıları açıklanmıştır. Ek olarak, FPGA kodlama esnasında kullanılan IP çekirdeklerden tez çalışmasında kullanılan kayan noktalı gösterim IP çekirdeği anlatılmıştır.

Çalışmanın beşinci bölümünde eğitimin gerçekleştirilmesi ve derin öğrenme metodunun ve parametrelerinin seçiminden bahsedilmiş, ardından tasarlanan FPGA mimarisi için hazırlanan altyapı ve mimari anlatılarak görsellerle desteklenmiştir. Yapılan çeşitli testler ve sonuçlar sunularak sonuçlar karşılaştırılmıştır.

Çalışmanın altıncı bölümünde çalışmaya ait sonuçlar değerlendirilmiş, çeşitli iyileştirme teknikleri hakkında önerilerde bulunulmuş ve gelecek çalışmalara katkıda bulunması amacıyla görüş ve önerilere yer verilmiştir.

# 2. LİTERATÜRDEKİ ÇALIŞMALAR

Baskılı devre üretimi sonrasında oluşan kusurların tespiti, elektronik kartın dahil edileceği cihazın verimliliği ve seri üretim sırasında oluşabilecek kusurlu cihaz sayısı açısından kritik önem taşımaktadır. Bu alanda uygulanan derin öğrenme teknikleri ve yazılım bazlı bilgisayarlı görü (CV) yöntemleri baskılı devre üzerindeki kusurları hızlı ve etkili bir şekilde tespit etmeye odaklanmaktadır. Özellikle derin öğrenme yöntemleri, küçük ve olağan düzene aykırı desenleri tespit edebilme kabiliyetleri ve yüksek miktarda veri setleri ile eğitildikleri taktirde yüksek doğruluk başarısına sahip olmalarıyla öne çıkmaktadır. Bu yöntemler, baskılı devre kusurları tespit sistemlerinin geliştirilmesinde son derece önemli bir rol oynamaktadırlar.

YOLO'nun farklı versiyonları günümüzde özellikle nesne tespiti konusunda yüksek derecede bir popülerliğe sahiptir. YOLO dışında RCNN (Recursive Convolutional Neural Network) gibi CNN (Convolutional Neural Network) tabanlı derin öğrenme yöntemleri de hala çalışmalarda kullanılmaktadır. Bu ağlar, öğrenme yetenekleri, öznitelik çıkarma ve gerçek zamanlı işlem yapabilme kabiliyetleri ile yüksek başarılar sağlayarak kendilerini kanıtlamışlardır. Çeşitli derin öğrenme metotları kullanılarak baskılı devre kusurlarının tespiti alanında başarılı sonuçlar elde edilmiştir.

Derin öğrenme algoritmalarını FPGA ile hızlandırarak performans artımı sağlamak da günümüzde uygulanan bir diğer etkili olarak kullanılan yöntemlerden birisidir. FPGA'nın herhangi bir ek çekirdeğe ihtiyaç duymadan direkt kodlama desteği ile paralel işlem yapabilme kabiliyeti ve tekrar programlanabilir olması, FPGA'i CPU ve GPU'ya göre çok daha hızlı kılan faktörlerden birkaçıdır. Derin öğrenme algoritmaları FPGA içerisine entegre edilerek, baskılı devre kusurlarının tespiti konusunda CPU ve GPU'ya göre kritik olmayan bir doğruluk kaybı ile daha yüksek hızlar elde edilebilmiştir. Böylece gerçek zamanlı sistemlerde derin öğrenme algoritmaları kullanımı daha kullanışlı hale gelmektedirler.

Literatürde FPGA tabanlı baskılı devre kusur tespiti ile ilgili herhangi bir derin öğrenme metodu kullanılmadan yapılan görüntü işleme içeren çalışmalar dahil birçok çalışma mevcuttur fakat özellikle YOLO'nun farklı versiyonları bu çalışmalarda oldukça yoğun bir şekilde kullanılmaktadır. Derin öğrenme yöntemleri FPGA'e entegre edileceği zaman FPGA içerisinde eğitilmemektedir fakat gerçek zamanlı entegrasyonu FPGA'de idame ettirilmektedir. Dolayısıyla eğitilmiş olan bu modeller derin öğrenme kabiliyetleri

sayesinde baskılı devrelerdeki en ufak kusurları dahi öğrenerek bu kusurların tespiti konusunda başarılı olmuş, FPGA ile çalıştırılarak ise tespit hızları kayda değer bir miktarda yükselmiştir.

Bu çalışmalarda kullanılan veri setleri genellikle açık kaynaklı veri setleri olup her çalışma kendine özgü bir yöntem ile bu veri setlerini çeşitlendirme yoluna gitmişlerdir. Bazı diğer çalışmalar ise kendi veri setlerini kendileri oluşturarak veri setlerinin performans ölçütlerini değerlendirmiş, ardından bu veri setiyle eğitilmiş derin öğrenme modelini direkt CPU veya GPU üzerinde çalıştırmışlar ya da FPGA içerisine uygulamasını gerçekleştirmişlerdir.

Bu çalışmalardan ilki, M. Aydın ve F. Kaçar'ın 2023 yılında gerçekleştirdiği herhangi bir derin öğrenme yöntemi kullanmadan baskılı devredeki eksik bileşenlerin tespitine yönelik bir çalışmadır [13]. OV7670 marka görüntü sensörü ile RGB555 renk formatında baskılı devre görüntüleri elde edilmiştir. ZedBoard geliştirme kartında bulunan Zynq-7000 FPGA'e ise bu görüntü RGB444 formatında ve 320x240 boyutunda verilmiştir. Çekim esnasındaki gürültüyü önlemek için LED aydınlatma halkası kullanmış, Medyan ve Gauss filtreleri de ek olarak kullanılmıştır. Ardından lehimin parlaklığını daha da ön plana çıkarıp tespit işleminim kolaylaşması için RGB görüntü HSV formatına çevrilmiştir. HSV renk uzayında piksellerin lehim rengini temsil eden gri tonlara yakın renk aralığı doygunluk ve değer hesaplanarak belirlenmiştir. Son olarak HSV formatında lehim renginin aralığı RGB formattaki kırmızı olarak uyarlanmıştır ve görüntü uzayı RGB formata geri döndürüldüğü zaman lehim olan alanlar kırmızı renk ile işaretlenmiştir.

YOLO algoritması kullanılmadan ele alınan bir diğer çalışma ise 2018 yılında E. Yuk ve arkadaşları tarafından gerçekleştirilen SURF algoritması ve rastgele ağaç (random forest) altyapısı kullanılarak dizgisi olmayan baskılı devrelerdeki kusurların tespitine yönelik yapılan çalışmadır [24]. Çalışma için kullanılan veri seti Kore'deki üretici bir firma tarafından tedarik edilmiştir. Veri setindeki görüntüler üzerinde SURF algoritması tabanlı öznitelik çıkarma yöntemi kullanılmıştır. SURF algoritması ile öznitelik çıkarımı yapılırken ilgi noktası (point of interest) Hesse Matrisi kullanılarak bulunmuş, çıkarım sonucunda ise her biri farklı bir özniteliği temsil eden 64 boyutlu bir tanımlayıcı oluşturmuşlardır. Çıkarılan özniteliklerin, baskılı devre görüntüsünde bulunan kenar, damla veya eğri gibi şablonlarla öznitelikleri temsil eden çok değişkenli vektörü kullanarak ağırlıklı çekirdek yoğunluğu tahmini (Weigted Kernel Density Estimation - WKDE) adı verilen rastgele ağaç algoritması

eğitilmiştir ve olasılıklar hesaplanmıştır. Performans testleri 10 adet görüntü üzerinde 10 katlı çapraz doğrulama kullanılarak yapılmış, tespit oranı %91 olarak elde edilmiştir. Yazarlar tarafından önerilen bu yöntemin baskılı devrelerdeki çiziklerin tespiti için uygun olduğu önerilmiştir.

Bir sonraki çalışma ise 2021 yılında J. Kim ve çalışma arkadaşları tarafından gerçekleştirilen Skip-Bağlantılı Evrişimsel Oto Enkoder kullanarak baskılı devredeki kusurların tespitine yönelik yapılan çalışmadır [25]. Bu çalışmada kullanılan veri seti W. Huang ve P. Wei tarafından 2019 yılında oluşturulan PCB-DATASET veri setidir [26]. Bu veri setindeki veriler büyük ebatta olduğundan görüntü üzerindeki kusurlu kısımlar 400x400 boyutlarında kırpılarak veri sayısı çoğaltılmış ve boyutları küçültülmüştür. Veri seti ön işlemeye tabi tutulmadan önce kontrast iyileştirmeleri için gri skalaya dönüştürülüp histogram eşitleme, ardından 3x3 boyutlarında bir çekirdek matrisi içeren medyan filtresi uygulanmıştır. Bu sayede veri setindeki dengesiz kısımlar ortadan kaldırılmıştır. Ön işleme sonrasında ise görüntü Skip-Bağlantılı Evrişimsel Oto Enkoder modeli NVIDIA GeForce RTX 2070 GPU desteği ile eğitilmiştir. Uygulama sırasında görüntü oto enkodere girdi olarak verilmiş ve tamir edilmiş bir görüntü üretilmiştir. Son olarak ise üretilen görüntü ile girdi görüntü birbirinden çıkartılıp elde kalan kısım kusur alanı olarak tespit edilmiştir. Eğitilmiş modele yapılan testler sonucunda tespit oranının %98'e kadar yükseldiği görülmüştür. Aynı zamanda hız olarak RTX 2070 platformunda çalıştırılan ve 55 FPS hızına ulaşan YOLOv4'ten [27] daha hızlı olduğu ifade edilmiştir.

Baskılı devre kusurlarının tespiti alanında gerçekleştirilen bir diğer çalışma aynı zamanda PCB-DATASET veri setini oluşturan W. Huang ve P. Wei tarafından 2019 yılında yürütülmüş olan çalışmadır [26]. Kusur sınıflandırması için CNN kullanılmıştır. Daha fazla eğitim görüntüsü üretmek ve veri sayısını artırmak amacıyla, görüntüdeki kusurların konumu, mevcut koordinatlar üzerinde rastgele 5 piksel ila 10 piksel ofset eklenerek değiştirilmiştir. Baskılı devre veri kümesinden kırpılan her orijinal kusurlu görüntünün çözünürlüğü bir görüntüden diğerine değiştiğinden dolayı kusur verilerinin kullanımını kolaylaştırmak amacıyla tüm görüntüler 64x64 çözünürlüğe yeniden boyutlandırılmıştır. NVIDIA GTX1080Ti GPU ile gerçekleştirilen eğitim sonucunda %99,40 eğitim, %97,74 test başarısına ulaşılmıştır.

Bir sonraki çalışma ise iki farklı veri seti üzerinde iki farklı metot uygulayarak baskılı devrelerdeki kusurları tespit etmeye yönelik 2021 yılında X. Wu ve arkadaşları tarafından

gerçekleştirilen çalışmadır [28]. Veri setlerinden bir tanesi W. Huang ve P.Wei'nin 2019 yılında oluşturduğu PCB-DATASET veri seti, diğeri ise S. Tang ve arkadaşları tarafından 2019 yılında oluşturulan ikilik tabanda bir veri seti olan DeepPCB veri setidir. Yöntemlerden birisi VGGNet (VGG16) altyapısı kullanan Tek Atış Çoklu Kutu Dedektörü (Single Shot Multibox Detector – SSD) model, diğeri ise resnet101 altyapısını kullanan bir Öznitelik Piramit Ağı (Feature Pyramid Network – FPN) [29] ile öznitelikleri çıkararak RPN destekli FRCNN [30] kullanıp işaretleme yapan bir modeldir. Alınan sonuçlarda ise SSD'nin doğruluk değerleri PCB-DATASET ve DeepPCB veri setlerinde sırasıyla %98,9 ve %89,7, F1-skor değerleri %70,2 ve %83,7 olarak elde edilirken FPN'in doğruluk değerleri %96,4 ve %92,7, F1-skor değerleri ise %97,3 ve %94,6 olarak bulunmuştur.

Hem bir veri seti hem de bir model sunan bir diğer çalışma ise DeepPCB veri setini oluşturan S. Tang ve arkadaşlarının yapmış olduğu çalışmadır [31]. Kendi oluşturdukları ikilik tabanda bir veri seti olan DeepPCB veri setini omurga kısmına VGG16-Tiny veya ResNet18 yerleştirilerek öznitelik çıkartılmış ve ardından Grup Piramit Havuzlama (Group Pyramid Pooling – GPP) modülüne girdi olarak sağlayarak GPP çıktısını evrişim modülleri ile birleştirip kusur tespiti yapan bir model sunmuşlardır. Modelin eğitimi Titan X GPU ile gerçekleştirilmiştir. Testlerin sonucunda önerilen yöntem, %98,6 eğitim doğruluğuna, %98,2 doğruluğa ve 62 FPS'lik bir kare hızına sahiptir.

Yapılan bir diğer çalışma ise M. Shen ve araştırma arkadaşları tarafından 2024 yılında gerçekleştirilen, YOLOv5 [32] tabanlı üretilmiş ve dizgisi yapılmış baskılı devre kartlarındaki hataların tespiti için PCBA-YOLO isimli bir derin öğrenme modeli sunan bir çalışmadır [33]. Çalışma dahilinde PCBA-DET ismi verilen 640x640x3 boyutlu görüntülere sahip bir veri seti oluşturulmuştur. Oluşturulan bu veri seti boyun kısmı uzamsal piramit havuzlama (SPPF) modülü ile değiştirilmiş, omurga kısmına ise büyük çekirdek evrişimi (kernel convolution) eklenerek modifiye edilmiş bir YOLOv5 modeli CUDA11.3 yazılımı destekli NVIDIA GeForce RTX 3080 ekran kartıyla Python 3.8 ortamında eğitilmiştir. PCBA-DET veri seti üzerinde değerlendirildiğinde ortalama hassasiyet %97,3, tespit algoritmasının gerçek zamanlı çalışma hızı 322,6 çerçeve/saniye (FPS) olarak elde edilmiştir.

A. Salunke Purva ve arkadaşları tarafından 2021 yılında yapılan bir çalışmada bir dijital kamera kullanarak baskılı devrelerin görüntülerini gerçek zamanlı olmayan bir şekilde yakalayan bir sistem geliştirmiştir [34]. Yakalanan görüntü, gri tonlamalı bir şablon

görüntüsü olarak bir veri tabanında saklanmıştır. Test görüntüleri, şablon eşleştirme için YOLO algoritması kullanılarak şablon görüntüsüyle karşılaştırılmıştır. YOLO algoritması yardımı ile eksik bölümler, kutuplar, devre kesintileri ve eksik izler gibi kusurlar ayırt edilmiştir.

A. Adibhatla ve arkadaşları tarafından 2021 yılında gerçekleştirilen bir çalışmada ise dört farklı kameranın panel görüntüsünün RGB formatında olmasını sağlayan bir AVI makinesinden toplanarak düzenlenen bir veri seti oluşturarak bir çalışma gerçekleştirilmiştir [35]. Çalışma dahilinde AVI makinesinden toplanan görüntüler 400x400 boyutlarda olacak ve kusurlu bölgeyi içerecek şekilde kırpılmış ve 23.000 adet kusurlu baskılı devre görüntüsü elde edilmiştir. Görüntüler toplandıktan sonra baskılı devre görüntülerindeki kusurlar kalite kontrol mühendisleri tarafından DEFECT etiketi ile işaretlenmiştir. Tespit modeli olarak o zamanın en yeni versiyonu olan YOLOv5'in küçük, orta ve büyük versiyonları kullanılmıştır ve karşılaştırılmıştır. YOLOv5 küçük ortalama %97,52 doğruluğa sahipken, YOLOv5 orta %99,16 ve YOLOv5 büyük ise %99,74 doğruluğa sahiptir. Doğruluk dışında mAP ile karşılaştırma sonuçlarına göre YOLOv5 büyük %94 olarak elde edilmiş, YOLOv5 orta ve YOLOv5 küçük ise %84 ve %82 olarak tespit edilmiştir.

J. Park ve arkadaşları tarafından 2023 yılında yürütülmüş bir çalışmada bazı derin öğrenme modellerinin baskılı devre kusurlarının tespiti açısından analizi yapılmıştır [19]. Çalışma dahilinde birden fazla veri seti incelenerek modellerde kullanılacak veri seti olarak TDD-PCB [36] veri setini tercih etmişlerdir. Belirtilene göre bir görüntüyü anlamanın üç farklı yolu olduğuna değinerek bu yollar ve metotları şu şekilde izah etmişlerdir: Parçalı Görüntü Sınıflandırma (ResNet50 – çekirdek ile eğitim), Bütün Görüntüyü Anlama (Resnet50 – çekirdeksiz eğitim) ve Direkt Kusur Tespiti (YOLOv7). Farklı eğitim/test veri oranları ile veri seti eğitilmiş ve test edilmiştir. Alınan sonuçlarda ise bütün yöntemler için en iyi sonuçları %80 eğitim %20 test olarak ayrılan eğitim vermiştir. Üç yöntem arasındaki en iyi doğruluğu ise %98,8 doğruluk ile Parçalı Görüntü Sınıflandırma yöntemi vermiştir.

R. Ding ve arkadaşları tarafından 2019 yılında gerçekleştirilen, hem bir veri seti ortaya çıkartan hem de yeni bir eğitim ağı ve modeli sunarak başarılı sonuçlar elde eden bir çalışma gerçekleştirilmiştir [36]. Oluşturulan veri seti 600x600x3 boyutlara sahiptir. Eğitim için kullanılan ve modifiye edilen modelde FRCNN altyapısını kullanarak öznitelik çıkartan katmanı FPN'in öznitelik çıkartıcısı ile yer değiştirilerek TDD-Net isimli yeni bir model ortaya koyulmuştur. Model, NVIDIA GeForce GTX1080 ekran kartı üzerinde CUDA 8.0.44

desteği ile Pyhton 2.7.12 ortamında eğitilmiştir. Alınan sonuçlarda ise sunulan model FRCNN ve FPN gibi modellere göre daha üstün başarı sağlamış ve %98,9 doğruluk yüzdesi sağlamıştır.

X. Liao ve arkadaşları tarafından 2021 yılında hayata geçirilen bir çalışmada YOLOv4 altyapısının modifiye edilmiş bir modeli ve YOLOv4-MN3 adıyla önerilmiştir [37]. Öncelikle bir baskılı devre görüntü alma cihazı kullanılarak özel bir veri seti oluşturulmuştur. Veri seti 4024x3036x3 boyutlarında görüntüler içermektedir. Daha sonra, çıkıntılı veya kırık hat, dağınıklık, çizik, hat onarım hasarı, delik kaybı ve aşırı yağ doldurma olmak üzere en yaygın altı kusur kategorisini içeren, 2008 örnekten oluşan bir veri kümesi toplanmıştır. Ardından YOLOv4 alt yapısında yapılan değişikliler olarak daha az parametreye ve düşük hesaplama maliyetine sahip olan MobileNetV3 hafif ağı, CSPDarknet53 yerine omurga ağı olarak kullanılmış boyun ve tahmin ağlarındaki farklı aktivasyon fonksiyonlarının etkisi test edilerek karşılaştırılmış ve Mish aktivasyon fonksiyonu seçilmiştir. Deneysel sonuçlar, önerilen YOLOv4-MN3 modelinin, baskılı devre yüzey kusur tespiti açısından daha yüksek tespit doğruluğu, daha hızlı tespit süresi ve mevcut en iyi algılayıcılara kıyasla %98,64'lük bir doğruluk ile daha üstün performans göstermektedir.

A. Bhattacharya ve S. G. Cloutier tarafından 2022 yılında literatüre sunulan bir çalışmada ise AOI araçlarını kullanarak baskılı devre kusur tespitini iyileştirmek için eksiksiz bir çerçeve sunulması hedeflenmiştir [38]. Kullanılan veri seti HRIPCB [39] adında 1386 adet etiketli görüntü içermektedir ve 10 farklı baskılı devre kartı görüntüsüne dayanmaktadır. Sunulan yeni metot YOLOv5'in bir varyasyonu olacak şekilde tasarlanmıştır. Hem dönüşümler hem de evrişimsel ağların harmanlanması ile iki yapının de avantajlarını birleştirmektedir. Bu sayede, küresel bağımlılıklar ve yerel bilgiler etkili bir şekilde kullanılabilmektedir. Ayrıca HSV veri artırımı yapılarak görüntü boyutundan da tasarruf sağlanmıştır. Sunulan modelin eğitimi Tesla T4 GPU ile gerçekleştirilmiştir. Alınan sonuçlarda ise YOLOv5m orta modeli ile kıyaslandığında YOLOv5m modelinden 3 kat daha az parametre kullanarak %3,2'lik ortalama eğitim doğruluğunda artış sağlanmıştır. En belirgin fark olarak ise YOLOv5m fazlalık bakır tespitinde %91,3 mAP'e sahipken sunulan model %96,9 doğruluğuna sahiptir.

Makine öğrenmesi ve derin öğrenme algoritmalarını kullanarak baskılı devre kusurlarının tespiti hakkında bir çalışmayı da V. Kaya ve İ. Akgül 2022 yılında

gerçekleştirmiştir [40]. Çalışmada HOG [41], KNN [42], SVM [43] makine öğrenmesi algoritmaları ve YOLOv4 derin öğrenme algoritması kullanılmıştır. Veri seti olarak ise PCB-DATASET isimli veri seti kullanılmıştır. Bahsi geçen algoritmalar, bulut tabanlı Google Colaboratory (Colab, 2022) ortamında Keras ve TensorFlow kütüphaneleri kullanılarak Python programlama dilinde eğitilmiş ve test edilmiştir. Eğitilen algoritmalar; HOG ve KNN birleşimi, HOG ve SVM birleşimi ve YOLOv4 olarak üç adettir. Elde edilen sonuçlarda ise en yüksek başarıyı %74,62 ile YOLOv4 elde etmiştir.

Küçük ölçekli nesneleri tespit etmede YOLOv5'i geliştirmek amacıyla bir çalışma da 2023 yılında J. Lim ve arkadaşları tarafından gerçekleştirilmiştir [44]. YOLOv5'i geliştirmeye yönelik yapılan çalışma olarak ise birden fazla özellik füzyonu uygulayarak geliştirilmiş bir FPN önerilmiştir. Tespit edici olarak FPN kullanılmış, omurga olarak C3 modeli uygulanarak omurgadaki son katman olarak SPP olarak yerleştirilmiştir. Eğitim esnasında hata değerini hesaplamak için kullanılan fonksiyon modifiye edilerek daha etkili bir hale getirilmiştir. Veri seti olarak DeepPCB veri seti kullanılmıştır. Veri setindeki bütün görüntüler 608x608 boyutta olacak şekilde yeniden boyutlandırılmıştır. Eğitim ve test gibi deneyler için kullanılan yazılım ortamı PyTorch v1.7.1, CUDA 11.1 ve cuDNN v8.0.35 olarak belirlenmiştir. Donanım olarak ise NVIDIA GeForce RTX 3070 GPU kartı kullanılmıştır. Yapılan deneyler ve testlerin sonucunda ise önerilen model %99,17 eğitim mAP değerine ulaşmış, %77,53 tespit doğruluğu ve 90 FPS hız ile çalışma performansını göstermiştir.

FPGA ile hızlandırılmış bir baskılı devre kusur tespit çalışması H. Yu ve arkadaşları tarafından 2023 yılında gerçekleştirilmiştir [5]. FPGA ile hızlandırma uygulanan model olarak ise DarkNet53 içeren YOLOv3 [45] derin öğrenme modeli kullanılmıştır. Veri seti olarak ise DeepPCB veri seti kullanılmıştır. Eğitilmiş modelin entegre edildiği FPGA olarak ise CPU ve FPGA'nın bir arada olduğu SoC bir çip olan ve ARM Cortex-R5 işlemci içeren AMD-Xilinx tarafından üretilen XCZU5EV-2SFVC784I çipi kullanılmıştır. Entegrasyon esnasında Xilinx tarafından sağlanan sistem geliştirme ortamı, tasarım sürecinde Vitis AI niceleyici, Vitis AI derleyicisi, algoritma tasarımı için Vitis AI çalışma zamanı, Vivado, petalinux, Vitis ve donanım tasarımına yönelik diğer araçlar dahil olmak üzere kullanılmıştır. Yapılan testler sonucunda ise tek bir çerçevede bulunan baskılı devre görüntüsündeki kusurların tespiti 97.59 milisaniye olup elde edilen doğruluk %78,9 olarak tespit edilmiştir.

FPGA tabanlı baskılı devre kusur tespiti çalışmalarından bir tanesi de 2024 yılında Y. Pen ve arkadaşları tarafından gerçekleştirilmiştir [46]. Çalışmada kullanılan baskılı devre kusurlarını barındıran orijinal veri seti dört bin adet görüntü içermektedir fakat veri çoğaltma işleminden sonra veri sayısı sekiz bine çıkartılmıştır. Yapılan çalışmada derin öğrenme modeli olarak araştırmacılar tarafından YOLOx-Plus adı verilen YOLOx'in geliştirilmiş hali kullanılmıştır. YOLO'nun omurgası olarak DarkNet53 seçilmiştir. Modelin eğitimi NVIDIA GeForce RTX 3080Ti ekran kartı donanımı ile, Cuda v10.1 ve CUDNN v7.6.5 desteğiyle Python 3.9.1 ortamında gerçekleştirilmiştir. Eğitilmiş modelin entegrasyonu ise SoC tabanlı ZYNQ FPGA içeren PYNQ-Z2 kartına yapılmıştır. Uygulama kısmında kameradan alınan görüntü FPGA'nın işlemci tarafından alınıp AXI aracılığı ile FPGA tarafına aktarılmıştır. FPGA tarafı için gerekli PLD kodu ise Vivado HLS ise yazılmıştır. Alınan sonuçlarda YOLOx-Plus, eğitim sonuçları olarak doğruluk %93,2'ye ulaşmıştır; YOLOx-Plus'ın FPGA'daki doğruluğu da %90'ın üzerine ulaşarak ağ kaybı 1,036 oranında azalmıştır. Algılama hızı 72,6 FPS olarak tespit edilmiştir.

FPGA ve YOLO tabanlı yapılan en güncel çalışmalardan bir tanesi 2025 yılında İ. Budak tarafından gerçekleştirilen YOLOv8 kullanarak FPGA tabanlı bir baskılı devre kusur tespiti yapan derin öğrenme uygulamasıdır [47]. Çalışmada kullanılan veri seti DeepPCB veri setidir. Yapılan çalışmada YOLOv4 ve YOLOv8 performansları kıyaslanarak YOLOv8'in daha iyi sonuçlar verdiği gösterilmiş ve YOLOv8m derin öğrenme modeli eğitilerek FPGA'ya uygulaması gerçekleştirilmiştir. Uygulama aşamasında FPGA olarak Kria KV260 SoC kullanılmış ve model sadece yazılım olarak sisteme entegre edilmiştir. Alınan sonuçlarda YOLOv8 modelinin eğitim mAP değeri %98,99 olarak belirtilmiş ve entegrasyon faaliyeti sonucunda bu değerin %98,21 olarak korunabildiği söylenmiştir. Tablo 2.1.'de literatürdeki çalışmalar özetlenmiştir.

**Tablo 2.1.** Literatür Çalışmaları

| Makale | Yazarlar | Yıl | Veri Seti | Kullanılan Cihaz | Yöntem | Performans (Doğruluk ve Hız) |
|---|---|---|---|---|---|---|
| Detection of printed circuit board faults with FPGA-based real-time image processing | Merve Aydın<br>Fırat Kaçar | 2023 | Yok | Zynq7000 | HSV Görüntü İşleme | Yüzdesel Değerlendirilmemiş |

**Tablo 2.1.** Devam Ediyor.

| Makale | Yazarlar | Yıl | Veri Seti | Kullanılan Cihaz | Yöntem | Performans (Doğruluk ve Hız) |
|---|---|---|---|---|---|---|
| Feature-Learning-Based Printed Circuit Board Inspection via Speeded-Up Robust Features and Random Forest | Eun Hye Yuk<br>Seung Hwan Park<br>Cheong-Sool Park<br>Jun-Geol Baek | 2018 | Özel Oluşturulmuş Veri Seti | Bahsedilmemiş | SURF + Rastgele Orman | %91 |
| Printed Circuit Board Defect Detection Using Deep Learning via A Skip-Connected Convolutional Autoencoder | Jungsuk Kim<br>Jungbeom Ko<br>Hojong Choi<br>Hyunchul Kim | 2021 | PCB-DATASET | NVIDIA GeForce RTX 2070 | Skip Bağlantılı Evrişimsel Oto Enkoder | %98<br>55 FPS |
| A PCB Dataset for Defects Detection and Classification | Weibo Huang<br>Peng Wei | 2019 | PCB-DATASET | NVIDIA GeForce GTX 1080Ti | CNN | %97,74 |
| PCB Defect Detection Using Deep Learning Methods | Xing Wu<br>Yuxi Ge<br>Qingfeng Zhang<br>Dali Zhang | 2021 | PCB-DATASET ve DeepPCB | Bahsedilmemiş | SSD ve FPN | SSD - %98,9<br>FPN - %96,4 |
| Online PCB Defect Detector On A New PCB Defect Dataset | Sanli Tang<br>Fan He<br>Xiaolin Huang<br>Jie Yang | 2019 | DeepPCB | Titan X | GPP | %98,6<br>62 FPS |

**Tablo 2.1.** Devam Ediyor.

| Makale | Yazarlar | Yıl | Veri Seti | Kullanılan Cihaz | Yöntem | Performans (Doğruluk ve Hız) |
|---|---|---|---|---|---|---|
| Defect detection of printed circuit board assembly based on YOLOv5 | Minghui Shen<br>Yujie Liu<br>Jing Chen<br>Kangqi Ye<br>Heyuan Gao<br>Jie Che<br>Qingyang Wang<br>Hao He<br>Jian Liu<br>Yan Wang<br>Ye Jian | 2024 | PCBA-DET | NVIDIA GeForce RTX3080 | YOLOv5s | %97,3<br>322,6 FPS |
| PCB (Printed Circuit Board) Fault Detection Using Machine Learning | Salunke Purva A.<br>Sherkhar Shubhangi N.<br>Arya Chandrapal S. | 2021 | Özel Oluşturulmuş Veri Seti | PC | YOLO | Bahsedilmemiş |
| Applying deep learning to defect detection in printed circuit boards via a newest model of you-only-look-once | Venkat Anil Adibhatla<br>Huan-Chuang Chih<br>Chi-Chang Hsu<br>Joseph Cheng<br>Maysam F. Abbod<br>Jiann-Shing Shieh | 2021 | Özel Oluşturulmuş Veri Seti | NVIDIA TITAN V | YOLOv5 | %94 |
| Analysis of Training Deep Learning Models for PCB Defect Detection | Joon-Hyung Park<br>Yeong-Seok Kim<br>Hwi Seo<br>Yeong-Jun Cho | 2023 | TDD-PCB | Bahsedilmemiş | YOLOv7 | %97 |
| TDD-Net: A Tiny Defect Detection Network for Printed Circuit Boards | Runwei Ding<br>Linhui Dai<br>Guangpeng Li<br>Hong Liu | 2019 | TDD-PCB | NVIDIA GeForce GTX 1080 | TDD-Net (Modifiye Edilmiş FRCNN) | %98,9 |

**Tablo 2.1.** Devam Ediyor.

| Makale | Yazarlar | Yıl | Veri Seti | Kullanılan Cihaz | Yöntem | Performans (Doğruluk ve Hız) |
|---|---|---|---|---|---|---|
| YOLOv4-MN3 for PCB Surface Defect Detection | Xinting Liao<br>Shengping Lv<br>Denghui Li<br>Yong Luo<br>Zichun Zhu<br>Cheng Jiang | 2021 | Özel Oluşturulmuş Veri Seti | NVIDIA GeForce RTX 3080 | YOLOv4-MN3 (Modifiye Edilmiş YOLOv4) | %98,64<br>56,98 FPS |
| End-to-end deep learning framework for printed circuit board manufacturing defect classification | Abhiroop Bhattacharya<br>Sylvain G. Cloutier | 2022 | HRIPCB | Tesla T4 Graphics | Modifiye Edilmiş YOLOv5 | %98,1 |
| Detection of Defects in Printed Circuit Boards with Machine Learning and Deep Learning Algorithms | Volkan Kaya<br>İsmail Akgül | 2022 | DeepPCB | Google Colab | YOLOv4<br>HOG + SVM<br>HOG + KNN | %74,62<br>%47,83<br>%39,86 |
| A deep context learning based PCB defect detection model with anomalous trend alarming system | JiaYou Lim<br>JunYi Lim<br>Vishnu Monn Baskaran<br>Xin Wang | 2023 | TDD-PCB | NVIDIA GeForce RTX3070 | YOLOv5 + FPN | %77,53<br>90 FPS |
| Design and implementation of embedded PCB defect detection system based on FPGA | Hongze Yu<br>Qingsong Lin<br>Cunling Liu | 2023 | DeepPCB | XCZU5EV-2SFVC784I (Zynq SoC) | YOLOv3 | %78,9<br>97,59ms algılama |
| Rapid Detection of PCB Defects Based on YOLOx-Plus and FPGA | Yajing Pan<br>Lei Zhang<br>Yujie Zhang | 2024 | Halka Açık Veri Seti | Xilinx PYNQ-Z2 (SoC) | YOLOx-Plus (Modifiye Edilmiş YOLOx) | %90,1<br>72,6 FPS |
| FPGA Tabanlı PCB Hata Tespitinde Derin Öğrenme Uygulaması | İzemnur Budak | 2025 | DeepPCB | AMD Xilinx Kria KV260 (SoC) | YOLOv8m | %98,99 eğitim<br>%98,21 uygulama |

# 3. KULLANILAN VERİ SETİ VE MAKİNE ÖĞRENMESİ

Bir makinenin öğrenmesi, bir hedefi anlama yolunda verilen sonuçların geliştirilmesine bağlı olarak gerçekleşen bir uyum sürecidir. Makine öğrenmesi (ML), geçmiş ve mevcut verileri analiz ederek geleceğe dair tahminlerde bulunmayı ve olası problemlere çözümler geliştirmeyi amaçlayan bir araştırma alanıdır. ML, bilgisayarların öğrenme sürecini destekleyecek algoritma ve tekniklerin oluşturulmasını sağlar. Başka bir ifadeyle, verilerden matematiksel ve istatistiksel yöntemlerle anlam çıkararak tahminler yapabilen sistemlerin oluşturulması sürecidir [48]. Günümüzde makine öğrenmesi başlığı altında birçok metot ve algoritmalar bulunmaktadır. Bu algoritmalar sayesinde makine öğrenmesinde kullanılan yazılımlar yeniden programlama gerektirmeyerek istenilen sonuca belirli bir süre sonunda yakınlaşabilme yetisine sahiptirler. Makine öğreniminin temel amacı, girdi verilerini alıp istatistiksel analiz kullanarak bir çıktı tahmin edebilen ve yeni veriler geldikçe çıktıları güncelleyebilen algoritmalar oluşturmaktır [49]. Makine öğrenmesi algoritmaları üç amaç için kullanılır; kümeleme, sınıflandırma ve regresyon. Makine öğrenmesinin daha detaya inmiş ve bir alt kümesi olarak değerlendirilebilecek bir nokta da derin öğrenmedir. Makine öğrenmesinde genel olarak basit yapay sinir ağları (YSA), destek vektör makinaları ve karar ağaçları gibi algoritmalar kullanılırken derin öğrenme algoritmaları daha büyük verileri işleyip öğrenebilen daha gelişmiş yapay sinir ağları kullanarak sonuca yaklaşır. Derin öğrenme metotları, birçok uygulamada genellikle geleneksel makine öğrenmesi ve veri analizi yöntemlerinden fark edilir derecede yukarıda bir performans sergiler [50].

Makine öğrenmesinde hiper parametre [51,52,53]. Alvesson ve Davidsson 2006 yılında yayınladıkları bir çalışmada hiper parametrelerin ayarlanmasının genellikle makine öğrenimi algoritmasının seçiminden daha önemli olduğunu göstermişlerdir [54].

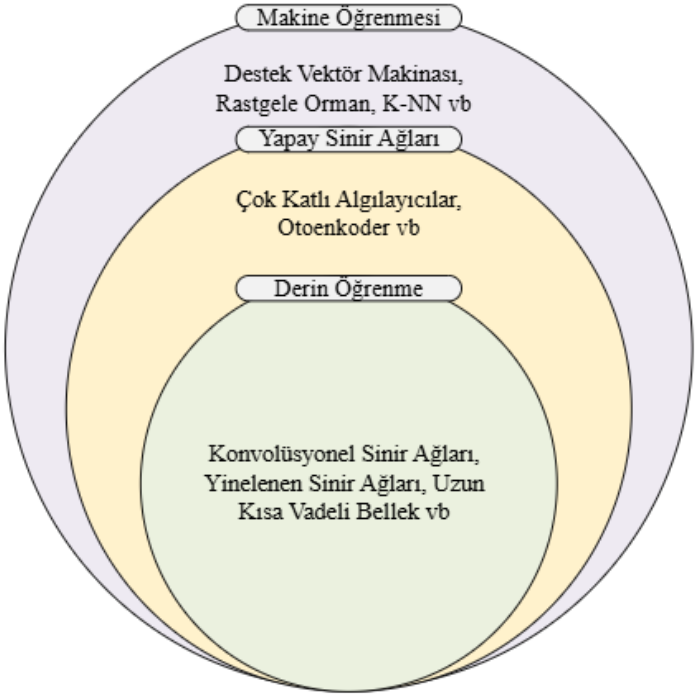
Makine öğrenmesi ve derin öğrenme algoritmalarının ulaşabileceği sonucun doğruluğu ve karmaşıklık seviyesi algoritmanın eğitildiği veri setinin büyüklüğü ve kalitesinden oldukça yüksek miktarda etkilenmektedir [55].

Makine öğrenmesinde karşılaşılan problemin çözümüne yönelik algoritma seçimi son derece önemli bir yere sahiptir. Her bir algoritma, veri setinin yapısına ve başarının hedefine göre farklı olumlu sonuçlar verir. Bu nedenle algoritmanın doğru seçilmesi eğitim ve çıkarım

sonuçlarında oldukça etkin bir rol oynamaktadır. Elde bulunan veri seti için yanlış bir algoritma seçimi algoritmanın bulunduğu modelin sonuçlarının başarısını düşürme ihtimaline sahiptir [56]. Algoritmaların sonuçlarının doğruluk değerlendirmesi genellikle tahmin doğruluğuna dayanır. Doğrulur değerlendirme formülü Eşitlik 3.1 ile ölçülebilir [57].

$$Doğruluk = \frac{Doğru\ Sınıflandırma\ Sayısı}{Toplam\ Test\ Sayısı} \tag{3.1}$$

Makine öğrenmesi, yapay sinir ağları ve derin öğrenme kavramlarının birbirleri arasında nasıl kümelendiği Şekil 3.1.'de görülmektedir.



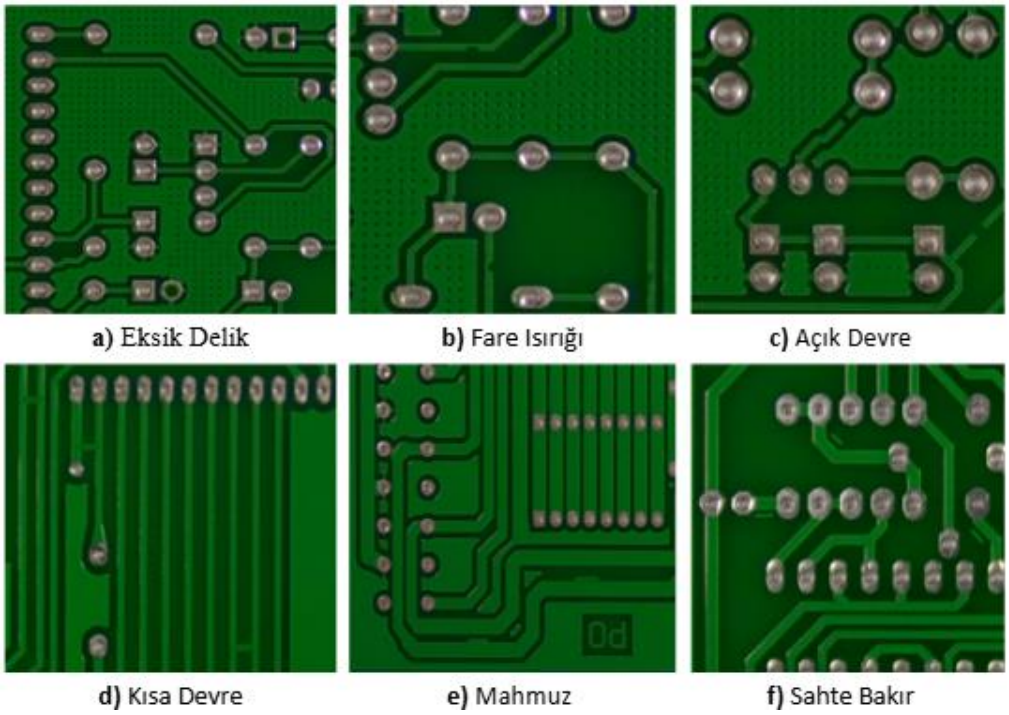

**Şekil 3.1.** Makine Öğrenmesi, YSA ve Derin Öğrenme Arasındaki İlişki

## 3.1. Veri Setleri

Derin öğrenme uygulamalarında veri setinin kalitesi ve çeşitliliği oldukça önemli bir yere sahiptir. Doğru veri setini seçmek eğitilen modelin performansını ve gerçek hayata uygulanabilirlik derecesini artırabilmektedir. Baskılı devrelerdeki kusurların tespitine yönelik açık kaynaklı birçok veri seti yer almaktadır. Literatür çalışmalarında elde edilen gözleme göre çalışmalarda genellikle halka açık veri setleri kullanılmaktadır. Bu çalışmada kullanılması adına birden çok veri seti araştırılmış, karşılaştırılmış ve tek bir veri setinde karar kılınmıştır.

### 3.1.1. PCB-DATASET

W. Huang ve P. Wei tarafından 2019 yılında oluşturulmuş bir veri setidir [26]. Veri seti 2000x2000x3 boyutunda 1386 adet renkli görüntüler içermektedir ve etiketlenerek sınıflara ayrılmış kusur sayısı 6 tanedir. Veri seti her bir görüntüde bir tip kusur sınıfı olacak şekilde oluşturulmuştur. Bu sınıflar şu şekildedir; eksik delik (missing hole), fare ısırığı (mouse bite), açık devre (open circuit), kısa devre (short), mahmuz (spur) ve sahte bakır (spurious copper). PCB-DATASET veri seti her ne kadar kaliteli ve yüksek çözünürlüklü görüntüler içerse de bu görüntüleri artırmak için uygulanacak yöntemler limitli ve belirli görüntülerde sabit kalacaktır. Bu yüzden veri sayısının kendi içeriğinde az oluşundan dolayı tercih edilmemiştir. Şekil 3.2.'de veri setindeki her bir kusura yönelik görüntülerden yakınlaştırılarak alınmış örnekler görülmektedir.

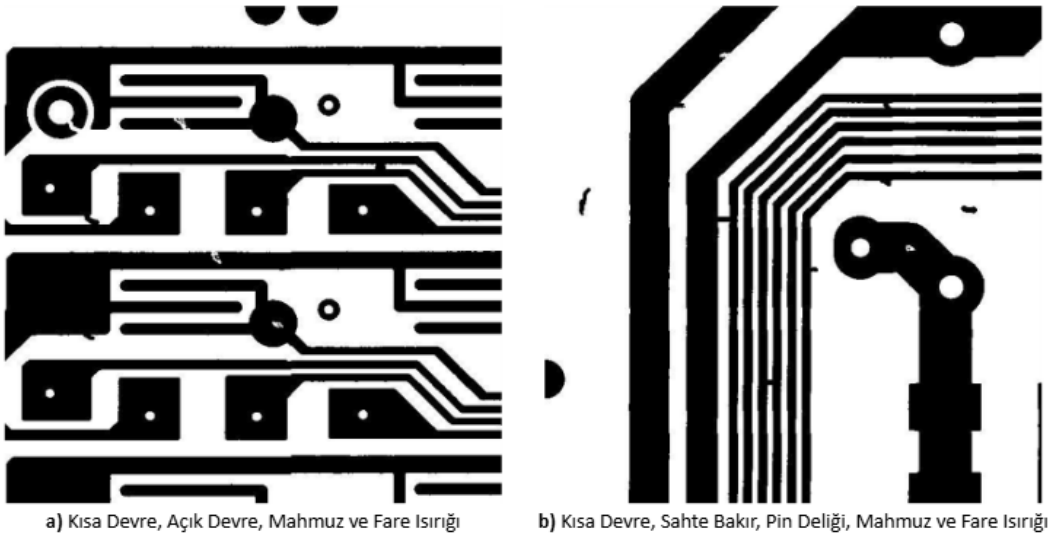



**Şekil 3.2.** PCB-DATASET Veri seti Görüntü Örnekleri; a) Eksik Delik, b) Fare Isırığı, c) Açık Devre, d) Kısa Devre, e) Mahmuz, f) Sahte Bakır

### 3.1.2. DeepPCB

S. Tang ve arkadaşları tarafından 2019 yılında oluşturulmuş bir veri setidir [31]. Veri seti 640x640 boyutlarında 1500 adet ikilik tabanda (binary) siyah-beyaz görüntü içermektedir. Veri setinde etiketlenmiş kusurlar açık (open), kapalı (short), fare ısırığı (mousebite), mahmuz (spur), pin deliği (pin hole) ve sahte bakır (spurious copper) olmak üzere 6 sınıftır. Görüntülerin içeriğinde bir görüntüde birden fazla kusur olacak şekildedir. DeepPCB veri setinde bulunan ikilik tabanlama işlemi, gürültü azaltma ve hesaplama

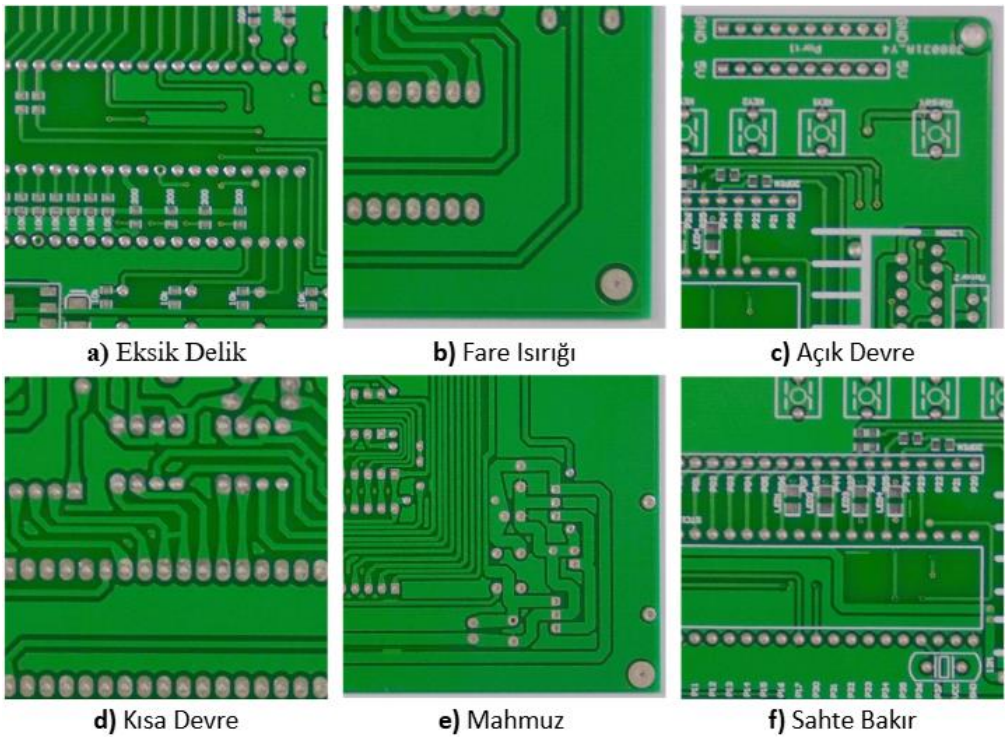
verimliliği sağlasa da ikilik tabandaki görüntüler ciddi anlamsal kayıplara uğrar. Ayrıca, yalnızca tek bir renk kanalına sahip olan ikili (binary) görüntüler, piksel düzeyinde anlam taşıyan bilgileri işleyen derin öğrenme algoritmalarının performansını olumsuz etkileyebilir. Buna karşılık, üç renk kanalına sahip olan standart RGB görüntüler, daha fazla özellik çıkarımı ve işleme imkanı sunar. Bundan dolayı DeepPCB veri seti tercih edilmemiştir. Şekil 3.3.'te DeepPCB veri setinde etiketlenmiş olan kusurlara ait görüntüler görülmektedir.




**Şekil 3.3.** DeepPCB Veri seti Görüntü Örnekleri; a) Kısa Devre, Açık Devre, Mahmuz ve Fare Isırığı ve b) Kısa Devre, Sahte Bakır, Pin Deliği, Mahmuz ve Fare Isırığı

### 3.1.3. TDD-PCB

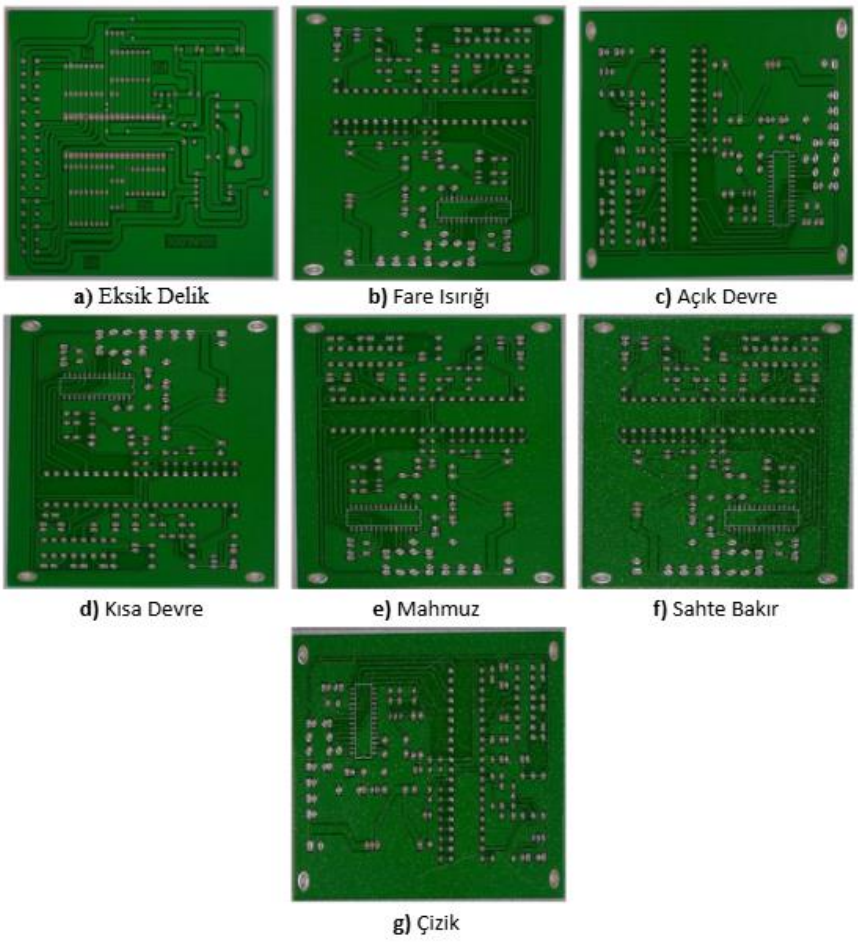
R. Ding ve arkadaşları tarafından 2019 yılında oluşturulmuş bir veri setidir [36]. Veri seti orijinalinde PCB-DATASET veri setinin 600x600x3 boyutlarında alt görüntülere kırpılmasıyla oluşturulmuştur. Kırpılan görüntüler 90, 180 ve 270 derece döndürülerek veri çoğaltma işlemi uygulanmıştır. Veri setinde test, doğrulama ve eğitim için toplamda 12428 adet görüntü bulunmaktadır. Veri setinde görüntü başına sadece bir çeşit kusur bulunmaktadır. TDD-PCB'de eksik delik (missing hole), fare ısırığı (mouse bite), açık devre (open circuit), kısa devre (short), mahmuz (spur) ve sahte bakır (spurious copper) olmak üzere toplam 6 adet etiketlenmiş sınıf bulunmaktadır. TDD-PCB veri seti kaliteli görüntüler içermektedir fakat veri setinin büyük bir çoğunluğu kalın ve birbirine çok yakın bakır yollar içeren görüntülerden oluşmaktadır. Gerçek uygulamalarda birden fazla kusurun bulunabilmesi ihtimalinden dolayı ve her bir görüntüde tek bir kusurun yer alması göz önünde bulundurularak bu veri seti tercih edilmemiştir. Şekil 3.4.'te TDD-PCB'ye ait her bir kusurun bulunduğu görüntüler bulunmaktadır.




**Şekil 3.4.** PCB-DATASET Veri seti Görüntü Örnekleri; a) Eksik Delik, b) Fare Isırığı, c) Açık Devre, d) Kısa Devre, e) Mahmuz, f) Sahte Bakır

### 3.1.4. PCBYOLO8

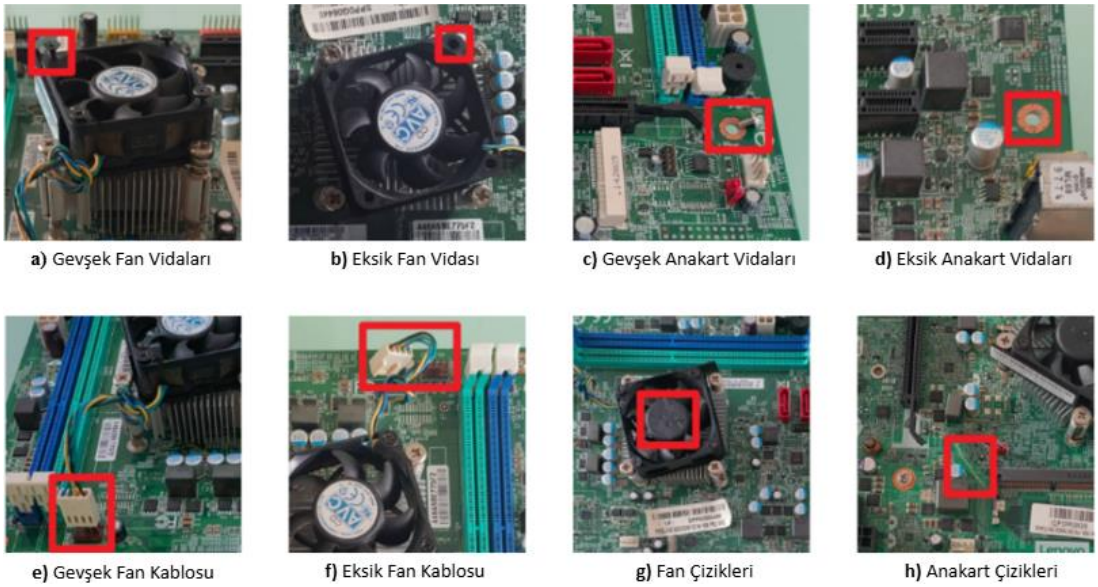
Roboflow ortamında PCBYOLO8 isimli bir kullanıcı tarafından oluşturulan PCBYOLO8 veri seti 2024 yılında oluşturulmuştur. Veri setinde 9761 adet 640x640x3 boyutunda diğer veri setleri görüntülerine göre daha bulanık görüntüler bulunmaktadır. Veri setinde veri çoğaltmak amacıyla görüntülere tuz ve biber gürültüsü eklenerek çeşitlilik artırılmıştır. PCBYOLO8'de eksik delik (missing hole), fare ısırığı (mouse bite), açık devre (opencircuit), kısa devre (shortcircuit), mahmuz (spur), çizik (scratch) ve sahte bakır (spurious copper) olmak üzere toplamda 7 adet etiketli sınıf bulunmaktadır. Her bir görüntüde 7 adet kusur sınıfından sadece bir adet yer almaktadır. PCBYOLO8'deki görüntülerin gürültü içeriyor olması bir avantajdır. Fakat veri setindeki görüntülerin düşük kalitede ve bulanık olması eğitilen modelin başarısını olumsuz etkileyecektir. Bu sebepten ötürü PCBYOLO8 tercih edilmemiştir. Veri setinde yer alan ve her bir kusuru barındıran görüntü örnekleri Şekil 3.5.'te görülmektedir.




**Şekil 3.5.** PCBYOLO8 veri seti Görüntü Örnekleri; a) Eksik Delik, b) Fare Isırığı, c) Açık Devre, d) Kısa Devre, e) Mahmuz, f) Sahte Bakır, g) Çizik

### 3.1.5. PCBA-DET

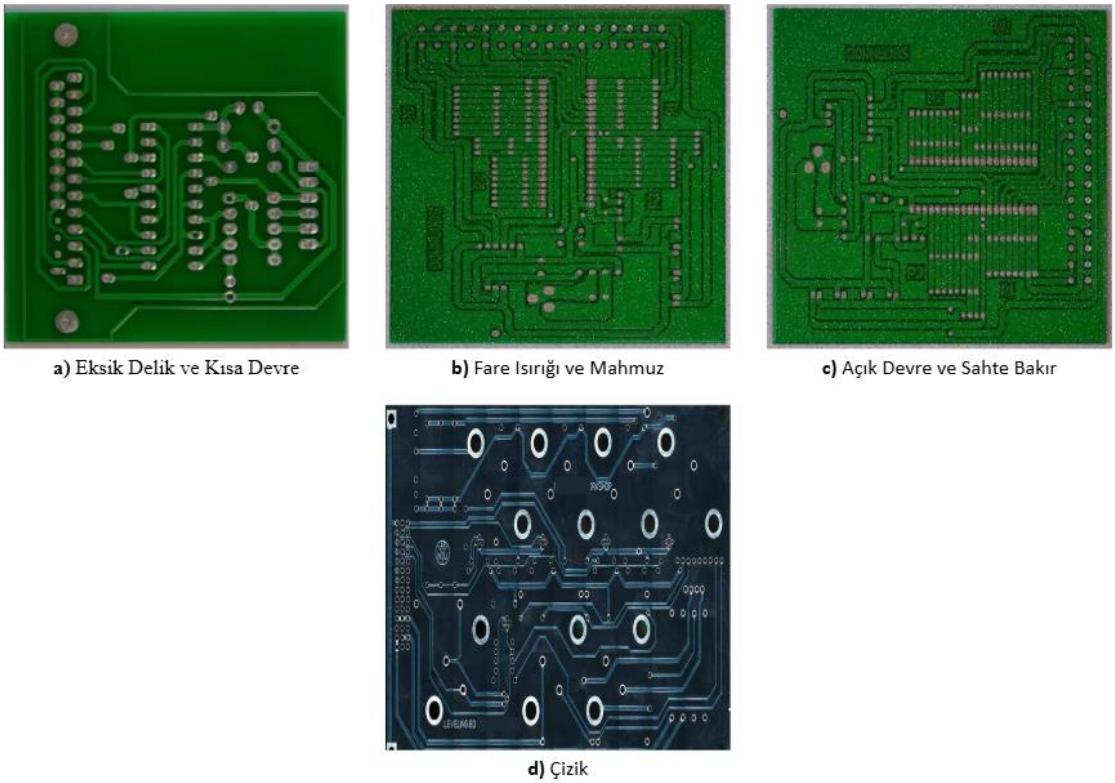
M. Shen ve arkadaşları tarafından 2024 yılında oluşturulmuş bir veri setidir. Veri seti baskılı devrelerden ziyade dizgisi yapılmış ve belli başlı kusurları olan kartları hedef alan bir veri setidir [33]. PCBA-DET'te 4000 adet 640x640x3 boyutlarında ve 8 adet kusur sınıfı etiketine sahip görüntüler bulunmaktadır. Bu 8 adet kusur sınıfı; Eksik Fan Vidası (Missing Fan Screw), Gevşek Fan Vidaları (Loose Fan Screws), Gevşek Anakart Vidaları (Loose Motherboard Screws), Eksik Anakart Vidaları (Missing Motherboard Screws), Gevşek Fan Kablosu (Loose Fan Wiring), Eksik Fan Kablosu (Missing Fan Wiring), Fan Çizikleri (Fan Scratches), Anakart Çizikleri (Motherboard Scratches) şeklindedir. PCBA-DET veri setinde bulunan görüntüler diğer veri setlerinden çok veri farklı ve veri çoğaltımı yapılması oldukça zor görüntülerdir. Ayrıca veri setindeki veri sayısının yetersiz olmasından dolayı bu veri seti tercih edilmemiştir. Şekil 3.6.'da her bir sınıfa ait veri seti görüntüsü bulunmaktadır.




**Şekil 3.6.** PCBA-DET Veri Seti Görüntü Örnekleri; a) Gevşek Fan Vidaları, b) Eksik Fan Vidası, c) Gevşek Anakart Vidaları, d) Eksik Anakart Vidalar, e) Gevşek Fan Kablosu, f) Eksik Fan Kablosu, g) Fan Çizikleri, h) Anakart Çizikleri

### 3.1.6. detection-pcb-defects-yolov8

Roboflow ortamında Bare PCB defects isimli kullanıcı tarafından 2024 yılında oluşturulan bir veri setidir [59]. Veri setinde 9666 adet 640x640x3 boyutunda, tuz-biber gürültüsü eklenerek ve görüntüler 90, 180 ve 270 derece döndürülerek veri çoğaltma işlemi yapılmış görüntüler bulunmaktadır. Veri seti, eksik delik (missing hole), fare ısırığı (mouse bite), açık devre (opencircuit), kısa devre (shortcircuit), mahmuz (spur), çizik (scratch) ve sahte bakır (spurious copper) olmak üzrere toplamda 7 adet etiketli sınıf içermektedir. Veri setindeki görüntülerde yalnızca bir sınıf baskılı devre kusurunun bulunduğu gibi birden farklı çeşit kusurun bulunduğu görüntüler de yer almaktadır. Bu veri setinde yer alan görüntülerin tek bir karta ait olmayışı, kendi içeriğinde görüntülerin döndürülerek, gürültü eklenerek ve bulanıklaştırılarak çoğaltılmış olması ve tek bir görüntüde birden fazla kusurun yer almasından dolayı, araştırılmış veri setlerinden daha avantajlı olduğuna karar verilmiş ve tez çalışmasında bu veri seti kullanılmıştır. Şekil 3.7.'de veri setinin içerisinde bulunan birkaç örnek sunulmuştur.



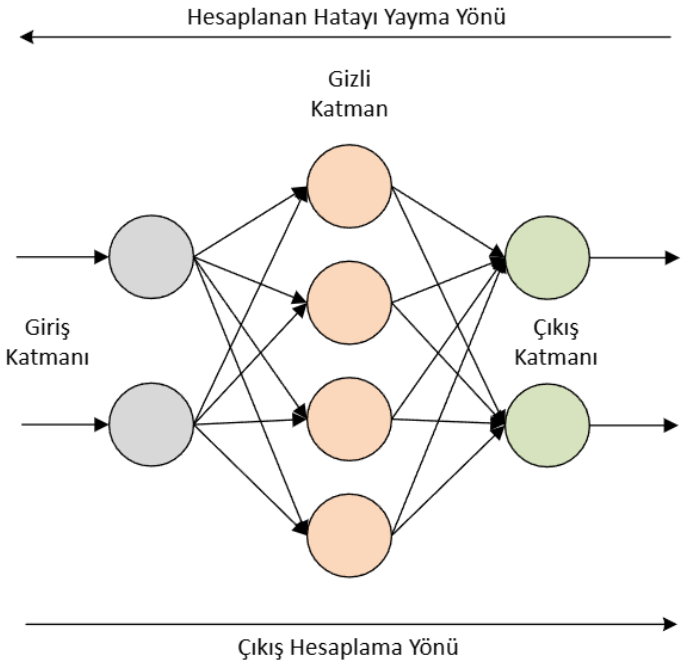

**Şekil 3.7.** detection-pcb-defects-yolov8 Veri Seti Görüntü Örnekleri; a) Eksik Delik ve Kısa Devre, b) Fare Isırığı ve Mahmuz, c) Açık Devre ve Sahte Bakır, d) Çizik

**Tablo 3.1.** Veri Setleri Özet Tablosu

| Veri Seti | Yıl | Veri Sayısı | Görüntü Boyutları | Sınıf İsimleri |
|---|---|---|---|---|
| PCB-DATASET | 2019 | 1386 | 2000 x 2000 x 3 | Missing Hole, Mouse Bite, Open Circuit, Short, Spur, Spurious Copper |
| DeepPCB | 2019 | 1500 | 640 x 640 | Open, Short, Mousebite, Spur, Pin Hole, Spurious Copper |
| TDD-PCB | 2019 | 12428 | 600 x 600 x 3 | Missing Hole, Mouse Bite, Open Circuit, Short, Spur, Spurious Copper |
| PCBYOLO8 | 2024 | 9761 | 640 x 640 x 3 | Falsecopper, Missinghole, Mousebite, Opencircuit, Pinhole, Scratch, Shortcircuit, Spur |
| PCBA-DET | 2024 | 4000 | 640 x 640 x 3 | Missing Fan Screw, Loose Fan Screws, Loose Motherboard Screws, Missing Motherboard Screws, Loose Fan Wiring, Missing Fan Wiring, Fan Scratches, Motherboard Scratches |
| detection-pcb-defects-yolov8 | 2024 | 9666 | 640 x 640 x 3 | Falsecopper, Missinghole, Mousebite, Opencircuit, Pinhole, Scratch, Shortcircuit, Spur |

## 3.2. Yapay Sinir Ağları

İnsan beyni, açıklaması ve çözümlemesi en zor yapıya sahip sistemlerden birisidir. İnsan beyninin işleyişinin incelenmesi ve matematiksel olarak modellenmesi ile Yapay Sinir Ağları (YSA) çalışma alanı meydana gelmiştir. YSA'da, her giriş verisine kendi ağırlığı ile çarpma işlemi uygulanmakta ve daha sonra toplama fonksiyonunda sonuçlar eşik değeri ile toplanmaktadır. Buradan çıkan sonucun aktivasyon fonksiyonunda işlenmesi ile çıkış bilgisi üretilmektedir. Aktivasyon fonksiyonu, sonucu doğrudan etkilemekte ve çıkış bilgisini gerekli sonlu aralıklarda sınırlandırmaktadır. Bu fonksiyonlar YSA uygulamalarında doğrusal veya doğrusal olmayan biçimlerde yaygın olarak tercih edilmektedir [60]. Şekil 3.8.'de basit bir YSA modeli bulunmaktadır ve Eşitlik 3.2'de ise bir YSA'daki bir adet nöronun çıkış formülü verilmiştir.




**Şekil 3.8.** YSA Model Örneği

$$ÇIKIŞ^{i}_{nöron} = \sum_{J=1}^{K_{giriş}} GİRİŞ^{i}\ x\ ağırlık^{iJ} + Eşik^{i} \qquad (3.2)$$

YSA günümüzde sıkça dile getirilse de ilk model günümüzden çok daha öncesine dayanmaktadır. 1943 yılında Warren McCulloch ve Walter Pitts tarafından ilk model biyolojik sinir sisteminden esinlenilerek geliştirilmiş ve ortaya koyulmuştur [61]. Bu gelişme ile makine öğrenmesinin de temelleri atılmaya başlanmış ve çalışmalarına ise 1950'lerde başlanmıştır. Bu alanın öncülerinden biri olan Alan Turing'in "Computing

Machinery and Intelligence" adlı makalesi, 1950 yılında yayımlayarak bu alanda önemli bir başlangıç yapmıştır [62]. Yapay sinir ağlarının tahmin etme ve öğrenme başarısını artıran geriye yayılım algoritması ise Plaut ve Hinton tarafından 1987 yılında literatüre sunulmuştur [63]. 1998 yılında ise Yann LeCun ve arkadaşları tarafından derin öğrenmenin temeli olan LeNet-5 adını verdikleri evrişimli sinir ağı ile rakamları sınıflandıran bir model geliştirilmiştir [64].

Bütün bu gelişmeler ve çalışmalar bir sonraki bölümde anlatılan derin öğrenmenin oluşmasında büyük rol almışlardır.

## 3.3. Derin Öğrenme

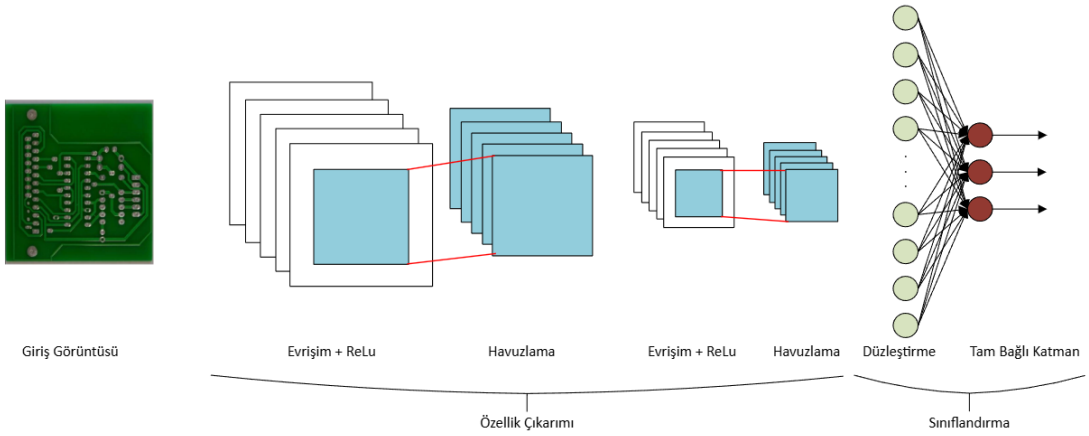
Birden fazla katmana sahip olan yapay sinir ağları modeli, derin öğrenme olarak isimlendirilir [65]. Derin öğrenme, sinir ağlarını oluşturarak ve eğiterek karmaşık problemlere çözüm sunan özel bir yaklaşımdır. Bu yöntem, bilgisayarların deneyimlerden öğrenmesini ve bilgiyi kavramsal hiyerarşi içinde işlemesini sağlar. Her kavram, daha temel kavramların birleşimiyle tanımlanır ve böylece sistem, insan operatörlerin tüm bilgileri açıkça belirtmesine gerek kalmadan yeni bilgiler türetebilir. Kavram hiyerarşisi sayesinde bilgisayar, daha basit yapı taşlarından yola çıkarak karmaşık kavramları öğrenebilir [66]. Derin öğrenme, yüksek doğrulukla problem çözme yeteneğiyle öne çıkar. Geleneksel makine öğrenme algoritmaları adım adım ilerlerken, derin öğrenme problemleri daha bütünsel bir perspektifle ele alarak kapsamlı çözümler üretir [67]. Derin öğrenme modelleri, özellik mühendisliğine ihtiyaç duymadan, verilerden otomatik olarak öğrenme yeteneğine sahiptir. Bu süreçte, evrişim işlemleri sayesinde farklı katmanlarda veri içindeki önemli ögeler yakalanır. Evrişimsel Sinir Ağları (CNN), bu mimarinin temel yapı taşı olup, görüntü sınıflandırma, nesne algılama ve görüntü segmentasyonu gibi alanlarda etkili çözümler sunar. Yapay sinir ağlarında gizli katmanların artırılmasıyla derinleşen CNN modelleri, 2 boyutlu filtreler aracılığıyla karmaşık veri yapılarını analiz edebilir. CNN'nin derin katmanları, giriş verisini farklı seviyelerde işleyerek anlamlı özellikler çıkarır. Modelin aşırı öğrenmesini önlemek için seyreltme (dropout) yöntemi kullanılır; bu teknik, eğitim sırasında rastgele bazı nöronları devre dışı bırakarak ezberlemeyi engeller ve modelin genelleştirme kapasitesini artırır [68].

## 3.4. Evrişimsel Sinir Ağları (CNN)

Derin öğrenmenin yeni bir teknoloji engelini aştığı zaman, genellikle evrişimli sinir ağları devreye girer. CNN (Convolutional Neural Networks) ya da ConvNets olarak bilinen

bu yapılar, derin sinir ağları modellerinde kullanılması en çok tercih edilen yöntemlerdir. Bu ağlar, bazı durumlarda, görüntüleri insanlardan daha etkili ve çok daha hızlı bir şekilde sınıflandırmayı öğrenebilirler [69].

1998 yılında Yann LeCun ve ekibi tarafından geliştirilen LeNet-5 mimarisi, CNN'lerin evrişimli ağlar arasında ilk başarılı uygulamalarından biri olarak kabul edilir. CNN, görsel korteksin bir özelliği olan reseptif alandan (receptive field) esinlenerek tasarlanmıştır. Bu ağ yapısı, hesaplama karmaşıklığını azaltırken bellek kullanımında verimlilik sağlar. Görüntülerden öznitelik çıkarma ve bu öznitelikleri çoğaltma becerisi, CNN'yi son derece yüksek sınıflandırma doğruluğuna sahip kılar. CNN'ler, eğim tabanlı öğrenmeyi azaltarak eğim sorununu daha iyi aşabilir. Eğim tabanlı bir algoritma tüm ağı izleyebilir ve bir hata kriterini en aza indirebilirse, CNN'lerin son derece iyi ayarlanmış bir ağırlıklandırma algoritması geliştirdiği bilinmektedir [70]. Şekil 3.9.'da genel bir CNN mimarisi görülmektedir.




**Şekil 3.9.** Genel CNN Mimarisi

Convolutional Layer (Konvolüsyon Katmanı), özellikleri saptamak için kullanılır. Non-Linearity Layer (Lineer Olmayan Katman), sisteme doğrusal olmamanın (non-linearity) tanıtıldığı katmandır. Pooling/Downsampling Layer (Havuzlama/İndirgeme Katmanı), ağırlık sayısını azaltır ve uygunluğu kontrol eder. Flattening Layer (Düzleştirme Katmanı), Klasik Sinir Ağı için verileri hazırlar. Tam Bağlı Katman, sınıflamada kullanılan Standart Sinir Ağıdır [71].

## 3.5. CNN Algoritmalarının Katmanları

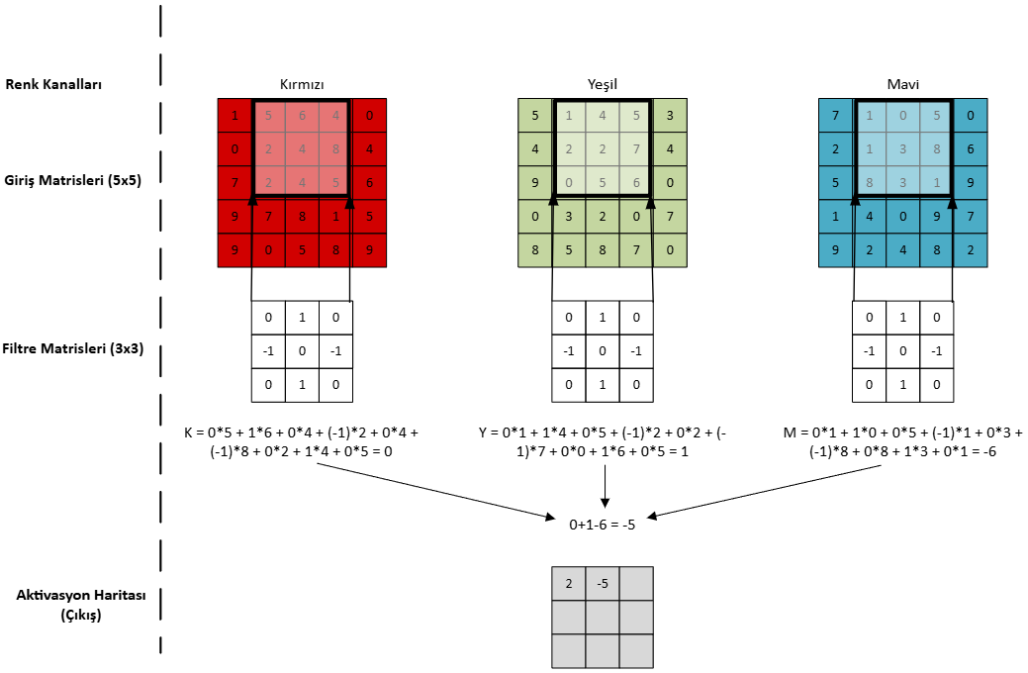
Evrişimsel sinir ağları 7 farklı katmandan oluşmaktadır. Bu katmanları hepsi zorunlu olmamak ile bazı katmanlar opsiyonel olarak kullanılmaktadır. Bu başlıkta bahsedilen katmanların açıklamaları yapılmıştır.

### 3.5.1. Giriş Katmanı (Input layer)

Bu katman, isminden de anlaşılacağı üzere CNN'nin ilk katmanını temsil eder ve burada veri, doğrudan ağa verilir. Modelin başarısı açısından bu katmandaki verinin boyutu kritik bir rol oynar. Giriş görüntüsünün boyutunun fazla olması, bellek kullanımını artırabilir, eğitim süresini uzatabilir ve test sürelerini de uzatabilir. Fakat, yüksek boyutlu giriş görüntüleri, ağın performansını da iyileştirebilir. Giriş boyutunun düşük olması ise bellek ihtiyacını azaltarak eğitim süresini kısaltabilir, ancak ağın derinliğini ve dolayısıyla performansını olumsuz etkileyebilir. Görüntü analizinde, ağ derinliği, donanım gereksinimleri ve ağın başarısını göz önünde bulundurarak uygun bir giriş görüntü boyutu seçilmelidir [72].

### 3.5.2. Evrişim Katmanı (Convolutional layer)

Evrişim katmanı CNN'nin temelini oluşturmasının yanı sıra dönüşüm katmanı olarak da bilinmektedir. Bu katman, veriye uygulanan konvolüsyon işlemi ile çalışmaktadır; burada belirli bir filtre, görüntü üzerinde kaydırılarak tüm resim boyunca geçmektedir. Filtreler, katmanlı mimarinin temel bileşenlerinden biridir ve genellikle 2x2, 3x3 veya 5x5 gibi farklı boyutlarda tasarlanabilir. Her bir filtre, bir önceki katmandan gelen görüntülere konvolüsyon işlemi uygulayarak çıkış verisini oluşturmaktadır. Bu işlem sırasında, her bir bölgedeki özellikler, aktif hale getirilir ve sonuç olarak bir aktivasyon haritası (ya da özellik haritası) elde edilerek belirli filtrelere ait özelliklerin keşfedildiği ve öne çıktığı alanları gösterir. Bu süreç, ağın özellikleri anlamasına ve doğru temsiller oluşturmasını mümkün kılar. CNN'nin eğitim sürecinde, bu filtrelerin katsayıları, her öğrenme adımında (iterasyonunda) sürekli olarak güncellenir ve optimize edilir. Eğitim verileri üzerinden yapılan bu güncellemeler sayesinde ağ, verinin hangi bölgelerinin önemli olduğunu belirleyerek, özelliklerin doğru şekilde tanımlanmasına katkı sağlar. Bu öğrenme süreci, modelin daha karmaşık temsiller oluşturabilmesini ve veriyi daha derinlemesine analiz edebilmesini sağlar. Aynı zamanda, ağın genelleme yeteneğini artırır ve gerçek dünya verileriyle daha iyi performans sergilemesine yardımcı olur. Bu katman, özellikle görüntü işleme gibi görevlerde son derece başarılı sonuçlar elde edilmesine olanak tanır [72].



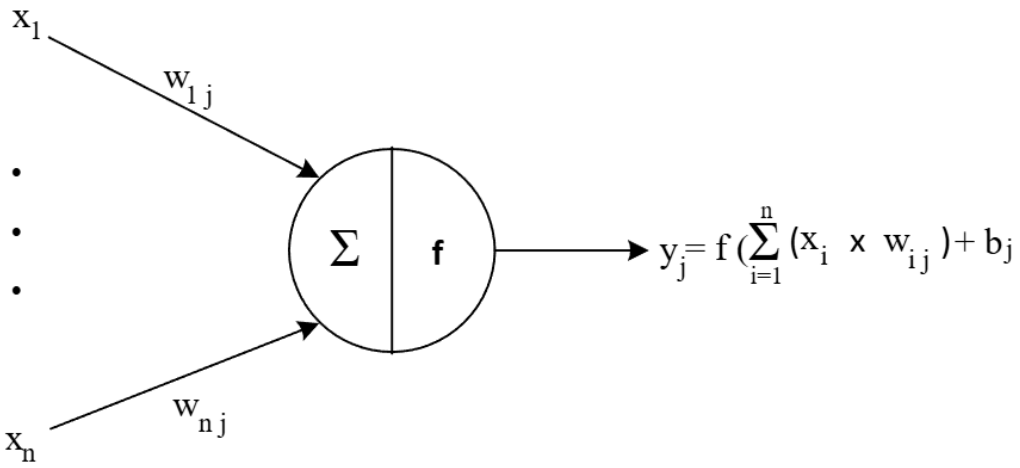

**Şekil 3.10.** 5x5x3 Boyutundaki Görüntüye 3x3'lük Filtre ile Konvolüsyon Örneği

Filtrelerin bir görüntü üzerindeki örnek uygulaması, Şekil 3.10.'da gösterilmektedir. Bu örneğe göre, giriş görüntüsünün renkli (RGB) olduğu ve 5x5 boyutlarında bir matris içerdiği kabul edilirse, veri boyutu 5x5x3 olarak belirlenir. Konvolüsyon işlemi, 3x3 boyutlarında bir filtre kullanılarak yapılır ve giriş görüntüsü, belirli bir adım (stride) ile sağa veya sola doğru kaydırılır. Filtre, görüntü üzerinde dolaşırken, matrisin kenarına ulaşıldığında bir satır aşağıya kayarak işlem devam eder. Bu kaydırma işlemi, görüntü matrisinin tamamı boyunca gerçekleştirilir. Her bir renk kanalındaki değerlerle filtre katsayıları çarpılır ve bu çarpımların toplamı alınarak çıkış değeri elde edilir. Aynı işlem, her üç renk kanalı için sırasıyla yapılır ve bu kanal verilerinin toplamı, aktivasyon haritasını oluşturur. Her renk kanalına uygulanan filtre katsayıları farklı olabilir ve tasarımcılar bu katsayıları modelin gereksinimlerine göre ayarlar. Aktivasyon haritasındaki değerler, giriş ve çıkış boyutları arasındaki benzer yoğunluk seviyelerini sağlamak amacıyla normalize edilerek her bir renk kanalı için hesaplanan değerin, o kanalın filtre katsayılarının toplamına bölünmesiyle yapılır. Bu tez çalışmasında kullanılan YOLOv11'in evrişim katmanlarında 3x3 ve 1x1 boyutlarında filtreler kullanılmaktadır.

### 3.5.3. Aktivasyon Fonksiyonu (Activation function)

Elde edilen özellik haritasında bulunan özelliklerin ayarlanmasında kullanılan fonksiyonlara aktivasyon fonksiyonu adı verilmektedir. Aktivasyon fonksiyonları, temelde yapay sinir ağı modellerinde doğrusal girdileri, doğrusal olmayan çıktılara dönüştürmek için

kullanılır. CNN, karmaşık özellikleri farklı şekilde ifade edebilmek adına fonksiyon formülünün sonucuna göre nöronun aktif olup olmayacağını bu aktivasyon fonksiyonu sayesinde belirlemektedir. Derin öğrenme modellerinde önemli bir özelliğe sahip olan bu katman da genelde ReLU (Rectified Linear Unit) aktivasyon fonksiyonu tercih edilmektedir [73]. CNN'de iki katman arasında bir aktivasyon fonksiyon bulunmaktadır. Bu aktivasyon fonksiyonunun nöron üzerinde bulunduğu nokta ve nöronun çıkışına etkisi Şekil 3.11.'de gösterilmektedir.

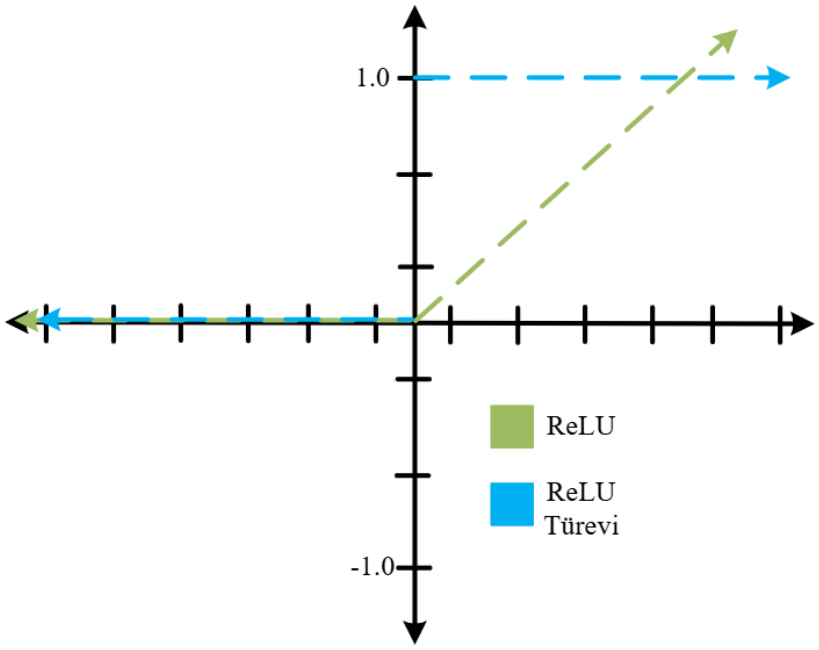



**Şekil 3.11.** Aktivasyon Fonksiyonu "f""in Nörondaki Yeri ve Çıkışa Etkisi

Şekil 3.11.'de de görüldüğü üzere $x_i$ girişi temsil etmektedir. n tane öznitelik j nöronuna aynı anda girdi olarak verilir. $w_{ij}$ ise girdiler ile nöron arasındaki ağırlıklardır. $b_j$, nöronun içerisindeki yanlılığı temsil etmektedir ve son olarak $y_j$, nöronun çıkış değeridir. Çıkış değeri bütün girdiler ile ağırlıkların çarpımının toplam değerinin aktivasyon fonksiyonu ile çarpılıp yanlılık değerinin eklenmesi ile bulunur. Günümüzde en çok kullanılan aktivasyon fonksiyonlarından bir tanesi olan ReLU'nun matematiksel ifadeleri Eşitlik 3.3 ve Eşitlik 3.4'te verilmiştir. Ayrıca ReLU fonksiyonunun kendisi ve türevinin grafiği de Şekil 3.12.'de gösterilmiştir [74, 75].

$$f(x) = \begin{cases} x, x \geq 0 \\ 0, x < 0 \end{cases} \tag{3.3}$$

$$f(x) = \max(x, 0) \tag{3.4}$$

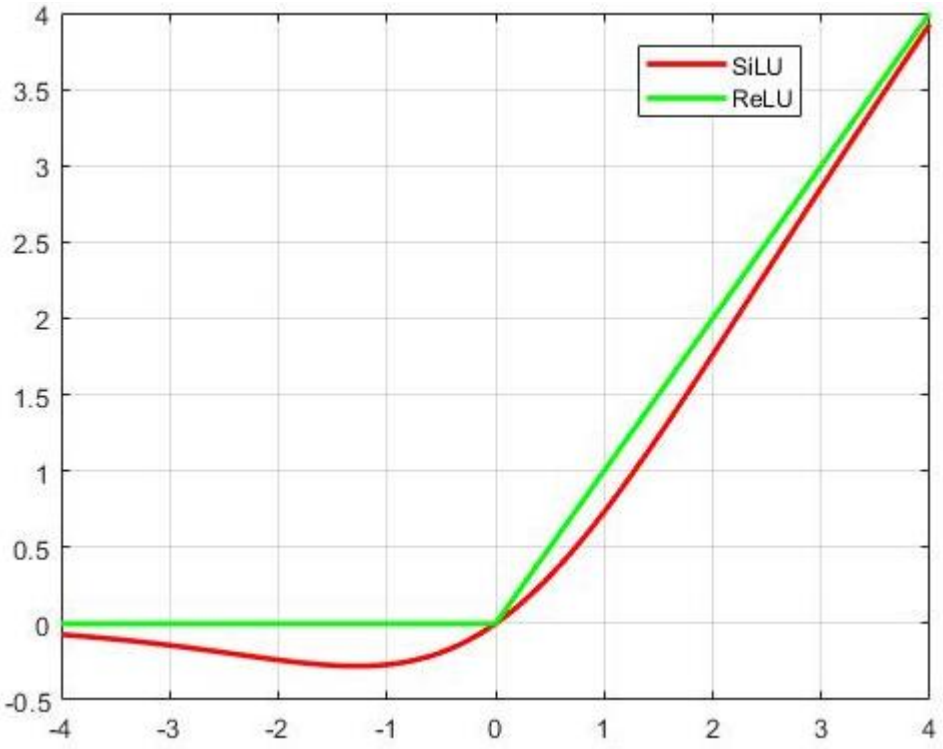



**Şekil 3.12.** ReLU Fonksiyonu ve Türevi Grafiği

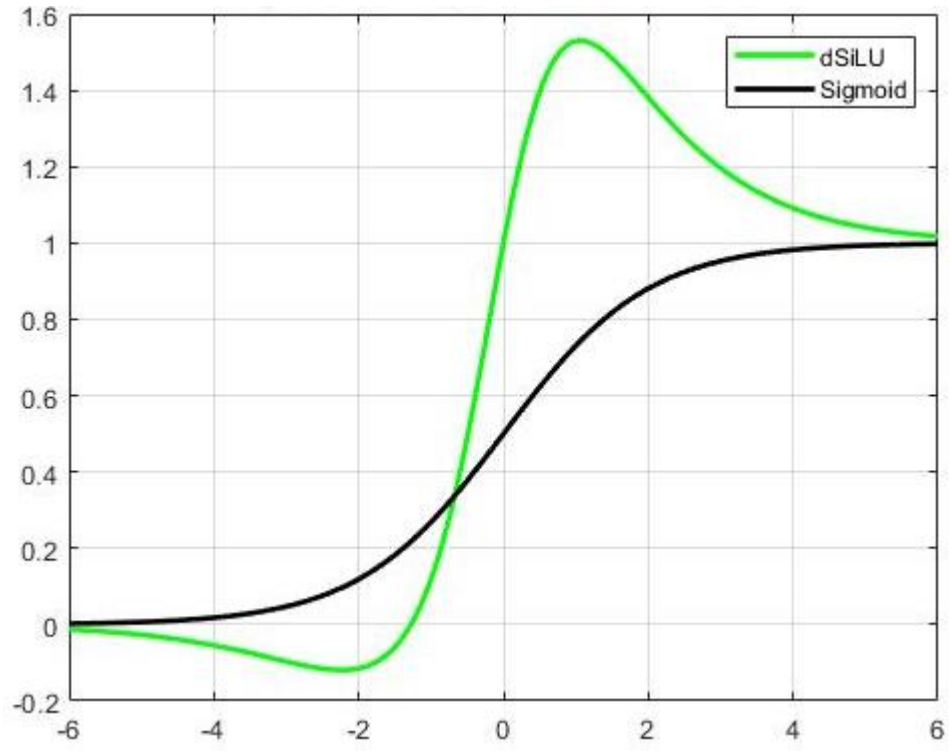
Bu tez çalışmasında kullanılan evrişimsel sinir ağlarının aktivasyon fonksiyonları Swish ya da Sigmoid Weighted Linear Unit (SiLU) aktivasyon fonksiyonudur. SiLU kullanılma sebebi ise en sık kullanılan fonksiyonlardan bir tanesi olan ReLU'dan görüntü alanında daha iyi sonuçlar vermiş olmasıdır. Fonksiyon ReLU ile çok benzer bir yapıya sahiptir. ReLU'nun (ve sigmoid ve tanh birimleri gibi yaygın olarak kullanılan diğer aktivasyon birimlerinin) aksine, SiLU'nun aktivasyonu monotonik olarak artmaz. SiLU'da türevin sıfır olduğu küresel minimum, büyük büyüklükteki ağırlıkların öğrenilmesini engelleyen örtük bir düzenleyici görevi gören ağırlıklar üzerinde bir "yumuşak taban" işlevi görür. Aynı zamanda SiLU'nun türev fonksiyonu, sigmoid fonksiyonunun daha dik ve ''aşırı atışlı'' bir versiyonu gibi görünmektedir. Eşitlik 3.4'te ve Eşitlik 3.5'te SiLU fonksiyonunun matematiksel ifadesi gösterilmiştir. Aynı zamanda SiLU fonksiyonu ile ReLU fonksiyonunun birlikte bulunduğu grafik verilmiştir Şekil 3.13.'te, SiLU'nun türevinin Sigmoid fonksiyonu ile kıyaslaması ise Şekil 3.14.'te verilmiştir [76].

$$f(x) = x \times sigmoid(x) \tag{3.4}$$

$$sigmoid(x) = \left(\frac{1}{1+ e^{-x}}\right) \tag{3.5}$$




**Şekil 3.13.** SiLU ve ReLU Fonksiyonları




**Şekil 3.14.** SiLU'nun Türevi ve Sigmoid Fonksiyonu

### 3.5.4. Havuzlama Katmanı (Pooling layer)

Bir Konvolüsyonel Sinir Ağı'nın (CNN) önemli bileşenlerinden biri de havuzlama (pooling) katmanıdır. Bu katman, ağın hesaplama yükünü ve parametre sayısını azaltmak amacıyla verinin uzamsal boyutunu kademeli olarak küçültür. Her bir özellik haritası (feature map) havuzlama katmanı tarafından ayrı ayrı işlenmektedir. En yaygın kullanılan yöntemlerden biri maksimum havuzlama (max pooling) tekniğidir. Uzamsal havuzlama (spatial pooling), ağın derinliğini azaltan doğrusal olmayan bir alt örnekleme yöntemidir ve genellikle maksimum veya ortalama havuzlama (average pooling) kullanılarak gerçekleştirilmektedir. Maksimum havuzlama, belirli bir filtre alanındaki en yüksek değeri seçerek bir sonraki katmana iletir ve böylece yalnızca en büyük değerlere sahip nöronlar elde tutulmaktadır. Öte yandan, ortalama havuzlama yönteminde ise filtre içerisindeki tüm nöronların değerleri dikkate alınarak bir ortalama hesaplanır ve bu değer bir sonraki katmana aktarılmaktadır [77, 78]. Bu tez çalışmasında yapılan havuzlama yöntemi 5x5 boyutlarında kernel matrisleri kullanılarak gerçekleştirilen maksimum havuzlama yöntemidir.

### 3.5.5. Tam Bağlı Katman (Fully connected layer)

Sınıflandırmadan önceki son katmandır, sınıflandırma skorlarının optimize edilmesinden sorumlu olan ve modelin çıktısını en iyi şekilde sınıflandırmak için softmax fonksiyonunu içeren bir yapıdır. Bu katman, öğrenme verileriyle yapılan işlemin nihai aşamasıdır. Softmax fonksiyonu, modelin çıktılarındaki her değeri bir olasılığa dönüştürerek, her sınıfın belirli bir veri örneğine ait olma olasılığını hesaplar. Bu hesaplama, genellikle çok sınıflı sınıflandırma problemlerinde, her bir sınıfın diğerlerine göre ne kadar olası olduğunu gösterir. Bu katman kendisinden önceki tüm katmanlardaki özelliklerin (feature) birleşimini kullanarak sınıflandırma kararlarını verir. Burada amaç, çıktıları olabildiğince doğru şekilde tahmin etmektir. Önceki katmanlar, veri üzerinde çeşitli özellikleri çıkararak soyutlamalar yapar ve bu soyutlamalar, son katmanda en iyi sonuçları elde etmek için birleştirilir. Bu katman, aslında bir tam bağlantı (fully connected) katmanı olup, her bir nöron, kendisinden önceki tüm nöronlarla bağlantılıdır ve bu sayede tüm bilgi optimizasyonu yapılır [79].

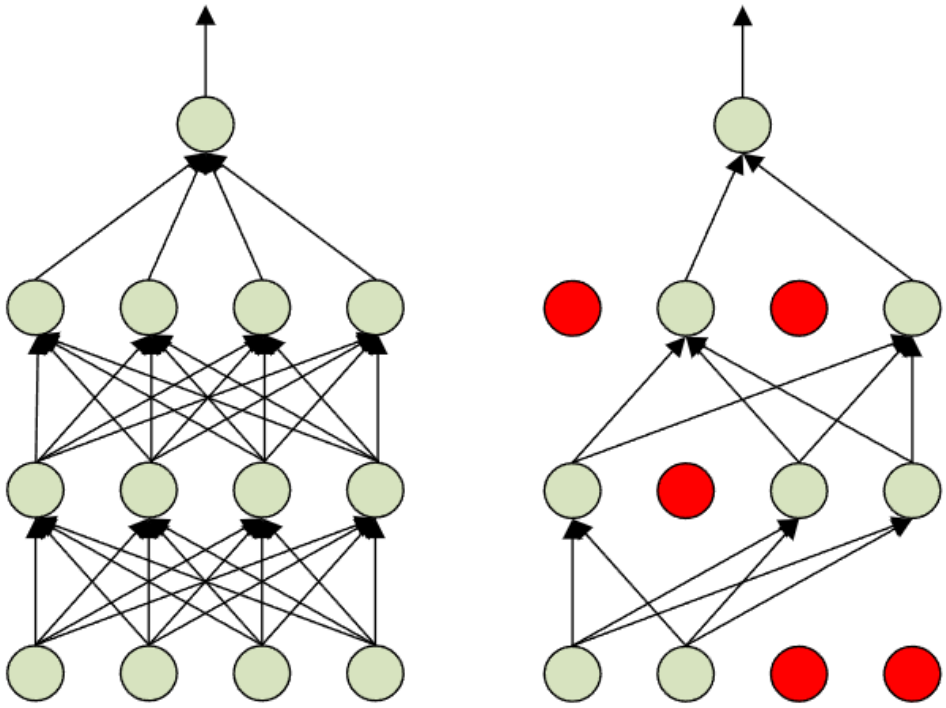
Tam bağlı (FC) katman genellikle en fazla parametreyi içeren sınıflandırma katmanından önce gelir, çünkü bu katmandaki her nöron kendisinden önceki katmandaki tüm nöronlara bağlıdır ve bu nöronlar arasındaki ilişkiye ilişkin parametreler çevrilir. Bu katmandaki girdi, karşılık gelen ağırlıklarla çarpılır, önyargılar eklenir ve doğrusal

olmayanlık evrişimsel katmanlar olarak tanıtılır [80]. Softmax'ın matematiksel modeli Eşitlik 3.6'da gösterilmiştir.

$$\sigma(z)_j = \frac{e^{z_j}}{\sum_{k=1}^{K} e^{z_k}} \tag{3.6}$$

### 3.5.6. Seyreltme Katmanı (Dropout layer)

Dropout, büyük veriler ile uğraşıldığında, ağın ezberlemesinin önüne geçmek için gizli katmanlar üzerinde, belirlenen ağırlıkların tekrar durumuna göre elemeler yapması işlemidir [81]. Eleme işlemi demek evrişimsel sinir ağı katmanlarında aşırı öğrenme ve aşırı uydurma olayının önüne geçme ve öğrenme olayın daha sağlıklı olması için kullanılan bir unutma katmanıdır. Ağın öğrenme yapmak yerine ezberlediği bilgilerin unutmasını sağlamak için kullanılır [82]. Seyreltme işleminde [0,1] aralığında bir eşik değeri kullanılarak zayıf bilgilerin unutulması sağlanır veya rastgele eleme yöntemiyle bazı nöronlar kaldırılır [83]. Standart bir CNN sinir ağı yapısı ile seyreltme işleminden sonra oluşan ağ yapısı Şekil 3.15.'te gösterilmiştir.



**Şekil 3.15.** Standart bir CNN ağına seyreltme işleminin uygulanması. a) Standart CNN ağ yapısı. b) Seyreltme katmanından sonraki ağ yapısı.

### 3.5.7. Sınıflandırma Katmanı (Classification layer)

Derin öğrenme modellerinde, tam bağlı (fully connected) katmandan sonra gelen sınıflandırma katmanı, ağın eğitim sürecinde öğrenilen özellikleri kullanarak son sınıflandırmayı yapmaktadır. Bu katmanın çıkış sayısı, sınıflandırılacak nesne sayısına doğrudan orantılıdır. Sınıf sayısı ile modelin çıkışındaki ağırlık matrisi sayısı aynı miktarda boyuta sahiplerdir. Tam bağlı katmandaki çıkış sayısı ile de bir adet ağırlık matrisinin boyutları aynıdır. Bu yapı, ağın öğrendiği özelliklerin doğru sınıflara yönlendirilmesini sağlamaktadır. Model, eğitim sırasında bu bağlantıları optimize eder ve her bir sınıf için en uygun tahminleri yapabilmek adına ağın çıktısını ayarlamaktadır [72].

Eğitim sürecinde, model bir dizi devir (epoch) boyunca nesnelerin farklı özelliklerini ayırt edebilme yeteneği kazanmaktadır. Her devir, ağın sınıflandırma kararlarını iyileştirebilmesi için öğrenme parametrelerinin güncellenmesini sağlamaktadır. Bu süreç, ağın her katmanında öğrenilen soyutlamaların giderek daha karmaşık hale gelmesiyle ilerlemektedir. Sonuç olarak, ağın çıkışı, giriş verilerine en uygun sınıfı tahmin etmeye yönlendirilmektedir.

Sınıflandırma işlemi sırasında, softmax fonksiyonu önemli bir rol oynamaktadır. Bu fonksiyon, çok sınıflı sınıflandırma problemlerinde her sınıfın olasılığını hesaplamak için kullanılmaktadır. Softmax, her bir çıkışı 0 ile 1 arasında bir değere dönüştürür ve bu değerler tüm sınıfların toplamı 1 olacak şekilde normalize edilmektedir. Bir başka deyişle, softmax çıkışındaki her bir sınıf için hesaplanan olasılıkların toplamı her zaman 1'e eşit olur. Bu çıkışlar, modelin her bir sınıf için tahmin ettiği olasılığı gösterir ve en yüksek olasılığa sahip sınıf, modelin nihai sınıflandırma kararı olarak kabul edilmektedir. Softmax fonksiyonu, modelin hangi sınıfın daha olası olduğuna karar vermesini sağlar ve bu karar, sınıflandırma işlemi için temel bir unsur oluşturarak modelin doğruluğu artırılır ve her sınıfın olasılığı daha doğru bir şekilde belirlenmesinde rol oynamaktadır.

## 3.6. You Only Look Once Version 11 (YOLOv11) Algoritma Katmanları

You-Only-Look-Once (YOLO), nesne tespiti için geliştirilen ve tek aşamalı çalışan bir derin öğrenme modelidir. Geleneksel yöntemlerin aksine, görüntüleri tek bir işlemle analiz ederek nesneleri tespit ederek sınıflandırır. Bu sayede, yüksek hız ve doğruluk sunarak gerçek zamanlı uygulamalarda büyük avantaj sağlamaktadır. YOLO modeli, görüntüdeki nesneleri belirlemek için bir evrişimli sinir ağı (CNN) içeren katmanlara sahiptir. Bu ağ, farklı boyutlardaki görüntüleri işleyerek nesnelerin temel özelliklerini çıkarmaktadır.

Görüntüdeki kenarlar, dokular ve şekiller gibi unsurlar analiz edilir ve bunlardan anlamlı bir bütün oluşturulur. Model, görüntü içerisindeki tüm nesneleri aynı anda algılayarak, geleneksel bölgesel tabanlı yöntemlere kıyasla çok daha hızlı sonuçlar üretmektedir [17].

Görüntüden çıkarılan özellikler daha sonra çeşitli ağ katmanları tarafından işlenerek nesne bilgileri zenginleştirilir. Bu aşama, modelin farklı ölçeklerdeki nesneleri daha iyi algılamasını sağlamaktadır. Küçük, orta ve büyük nesneler arasındaki farkları daha iyi ayırt edebilmek için özellik haritaları birleştirilir ve detaylandırılır. Böylece, model yalnızca nesnelerin varlığını belirlemekle kalmaz, aynı zamanda onların konumlarını ve sınırlarını da hassas bir şekilde belirlemektedir. Son aşamada, işlenen veriler kullanılarak nesnelerin sınıfı ve konumu tahmin edilmektedir. Model, her nesnenin hangi kategoriye ait olduğunu belirler ve sınır kutularının koordinatlarını oluşturmaktadır. YOLO'nun en büyük avantajlarından biri, bu süreci tek bir geçişte tamamlamasıdır. Böylece, görüntü işleme süresi minimuma iner ve model, gerçek zamanlı uygulamalar için ideal hale gelmektedir. Bu modelin sağladığı hız ve doğruluk, modelin birçok alanda kullanışlı olmasında büyük bir etmendir. Otonom araçlardan güvenlik kameralarına, tıbbi görüntü analizinden perakende sektörüne kadar geniş bir yelpazede uygulanmaktadır. YOLO, nesne tespiti konusunda çığır açan bir yaklaşım sunarak hem akademik araştırmalarda hem de endüstriyel uygulamalarda en çok tercih edilen modellerden biri olmuştur. YOLO ailesinin YOLOv11'e kadar olan model gelişimi Tablo 3.2.'de verilmiştir.

**Tablo 3.2.** YOLO Modellerinin Evrimi

| Versiyon | Yıl | Görevler | Katkılar | Altyapı |
|---|---|---|---|---|
| YOLO | 2015 | Nesne Algılama<br>Basit Sınıflandırma | Tek aşamalı nesne dedektörü | Darknet |
| YOLOv2 | 2016 | Nesne Algılama<br>Basit Sınıflandırma | Çok ölçekli eğitim ve boyut kümelemesi | Darknet |
| YOLOv3 | 2018 | Nesne Algılama<br>Çok Ölçekli Algılama | SPP bloğu ve Darknet-53 omurgası | Darknet |
| YOLOv4 | 2020 | Nesne Algılama<br>Temel Nesne Takibi | Mish aktivasyonu ve CSPDarknet-53 omurgası | Darknet |
| YOLOv5 | 2020 | Nesne Algılama<br>Temel Örnek Bölümlendirme (özel değişiklikler yoluyla) | Çapasız algılama, SWISH aktivasyonu ve PANet | PyTorch |
| YOLOv6 | 2022 | Nesne Algılama<br>Örnek Bölümlendirme | Öz-dikkat ve çapasız OD | PyTorch |

**Tablo 3.2.** Devam Ediyor.

| Versiyon | Yıl | Görevler | Katkılar | Altyapı |
|---|---|---|---|---|
| YOLOv7 | 2022 | Nesne Algılama<br>Nesne İzleme<br>Örnek Segmentasyonu | Transformatörler ve E-ELAN yeniden parametrelendirme | PyTorch |
| YOLOv8 | 2023 | Nesne Algılama<br>Örnek Segmentasyonu<br>Panoptic Segmentasyon<br>Anahtar Nokta Tahmini | GAN'lar ve çapasız algılama | PyTorch |
| YOLOv9 | 2024 | Nesne Algılama<br>Örnek Bölümlendirme | PGI ve GELAN | PyTorch |
| YOLOv10 | 2024 | Nesne Algılama | NMS'siz eğitim için tutarlı ikili atamalar | PyTorch |
| YOLOv11 | 2024 | Nesne Algılama<br>Örnek Bölümlendirme<br>Poz Tahmini<br>Yönlendirilmiş Nesne Algılama | Daha az parametre ile daha derin modül üretimi ve<br>Aşamalar Arası Kısmi + Mekansal Dikkat | PyTorch |

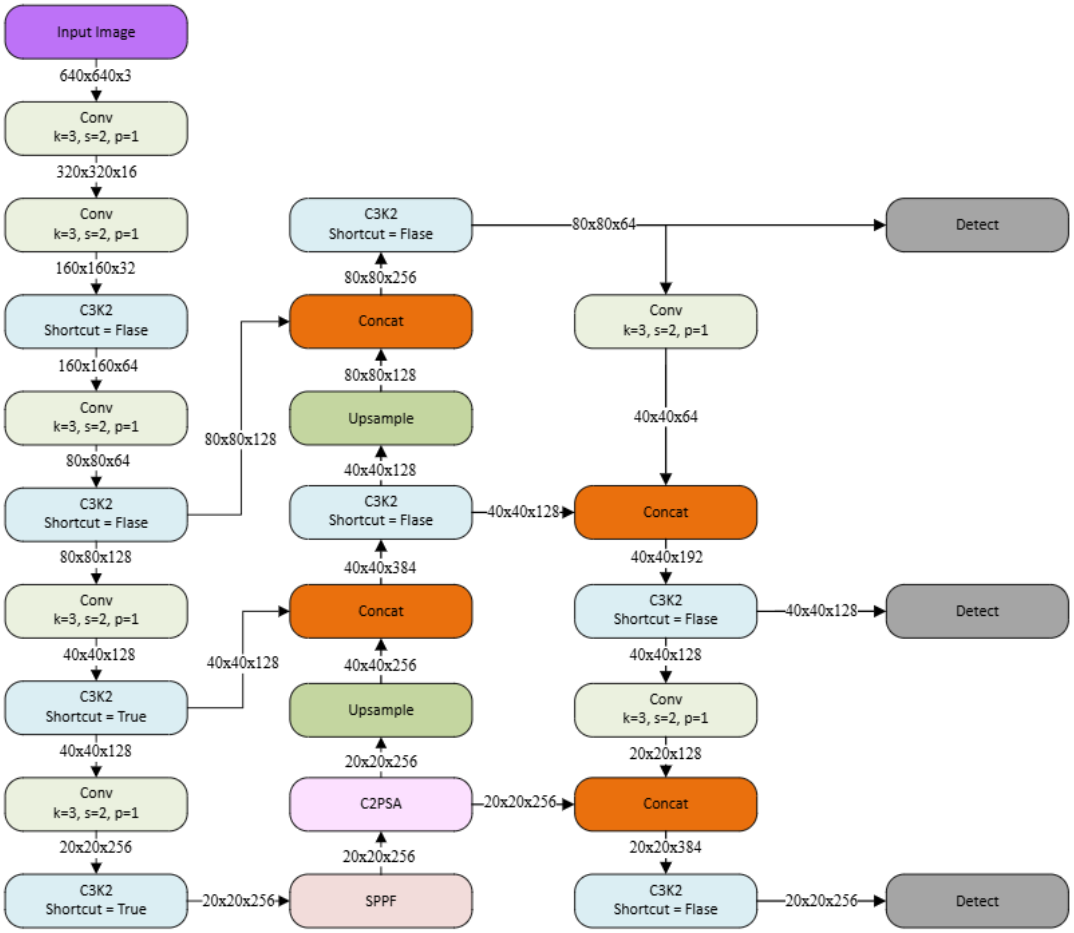
YOLO algoritmasının gelişimi, gerçek zamanlı nesne tespitinde çığır açan bir yenilik olan YOLOv11 ile yeni bir aşamaya ulaşmaktadır. Bu sürüm, önceki versiyonların güçlü yönlerini koruyarak performansını daha da iyileştirirken, aynı zamanda yenilikçi özellikler ekleyerek çeşitli bilgisayarla görme (CV) uygulamalarında kullanımını genişletmektedir. YOLOv11, gelişmiş mimarisi sayesinde daha yüksek doğruluk, hız ve verimlilik sunarak, nesne tespiti alanındaki zorlukların üstesinden gelmeyi hedeflemektedir. Model, yalnızca geleneksel nesne algılama görevlerinde değil, aynı zamanda duruş tahmini, örnek bölütleme ve diğer ileri düzey bilgisayarla görme uygulamalarında da etkili olacak şekilde tasarlanmıştır. YOLOv11, geliştirilmiş uyarlanabilirliği sayesinde geleneksel nesne tespitinin ötesine geçerek daha geniş bir bilgisayarla görme (CV) görev yelpazesini desteklemektedir. YOLOv11'in tasarımı, gücü ve pratikliği dengeleyerek çeşitli endüstrilerdeki belirli zorlukları daha yüksek doğruluk ve verimlilikle ele almayı amaçlamaktadır. Gelişmiş mimarisi ve optimize edilmiş hesaplama gereksinimleri sayesinde hem büyük ölçekli endüstriyel kullanımlar hem de düşük güçlü cihazlarda verimli bir şekilde çalışabilir [22].

YOLOv11, COCO veri kümesinde daha az parametre kullanarak daha iyi performans elde etmek amacıyla YOLOv8 temel alınarak optimize edilmiştir. Yeniden tasarlanan omurga (backbone) ve boyun (neck) yapısının yanı sıra yeni bir bileşen olan C3k2'nin eklenmesi sayesinde, YOLOv11'in görsellerden özellik çıkarma yeteneği önemli ölçüde

geliştirilmiştir. Bu iyileştirmeler, modelin çoklu hedef tespit görevlerinde daha da başarılı olmasını sağlamaktadır. Özellik çıkarma verimliliği, hedeflerin doğru bir şekilde konumlandırılması ve sınıflandırılması için kritik bir faktördür. YOLOv11'in yeni mimarisi, tespit hassasiyetini ve doğruluğunu büyük ölçüde artırmaktadır. Ayrıca, model daha etkili bir mimari ve eğitim prosedürü kullanmaktadır. Orijinal tespit başlığına (detection head) eklenen iki derin ayrılabilir evrişim (Depthwise Separable Convolution - DWConv) modülü, modelin parametre ve hesaplama gereksinimlerini önemli ölçüde azaltırken, yüksek doğruluğu koruyarak işlem hızını artırmaktadır [84].

Bu tez çalışmasında derin öğrenme modeli olarak YOLO versiyonları arasından YOLOv11 derin öğrenme modelinin "n" versiyonu (YOLOv11n) kullanılmıştır. Model olarak YOLOv11 varyantları arasında ana model iskeleti açısından herhangi bir değişiklik bulunmamaktadır. YOLOv11n derin öğrenme modelinin 640x640x3 boyutlarındaki bir girdi görüntüsü için ara katmanlardaki çıktı boyutlarını da içeren model görüntüsü Şekil 3.16.'da gösterilmektedir.

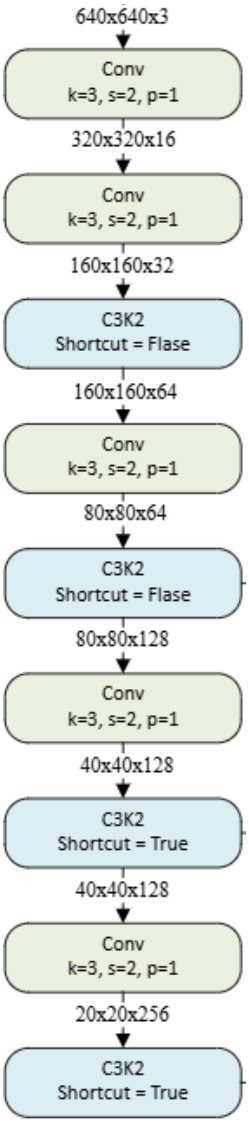
YOLOv11 modeli de diğer uçtan uca tespit sistemlerinin de genel olarak sahip olduğu gibi 3 ana parçadan oluşmaktadır; Omurga (Backbone), Boyun (Neck) ve Kafa (Head).




**Şekil 3.16.** YOLOv11 Derin Öğrenme Modeli

### 3.6.1. Omurga (Backbone)

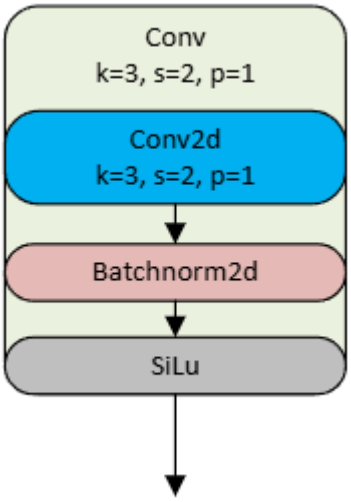
Omurga, modelin temel yapı taşı olup, görüntüden anlamlı özellikler çıkaran bir evrişimli sinir ağı (CNN) olarak çalışır. Model, farklı boyutlardaki giriş görüntülerini işleyebilir ve bunların içindeki nesnelerin temel özelliklerini belirlemektedir. Bu katman, görüntüdeki kenarlar, dokular, renk dağılımları gibi düşük seviyeli bilgileri öğrenerek daha derin katmanlarda nesneye özgü karmaşık desenler ve şekiller oluşturmaya yardımcı olmaktadır. Geleneksel olarak YOLO modelleri, Darknet, CSPDarknet veya ResNet gibi güçlü ve optimize edilmiş evrişimli sinir ağı mimarilerini omurga olarak kullanmaktadır. Bu bileşen, modelin öğrenme kapasitesini ve genel doğruluk oranını büyük ölçüde etkilemektedir [22, 85]. YOLOv11'in omurgası bileşenlerin bulunma yoğunluğuna göre iki ana başlığa ayrılabilir; evrişimsel katmanlar ve C3K2. YOLOv11'in omurga yapısı Şekil 3.17.'de gösterilmiştir.




**Şekil 3.17.** YOLOv11 Omurga (Backbone) Yapısı

#### 3.6.1.1. Evrişimsel Katmanlar

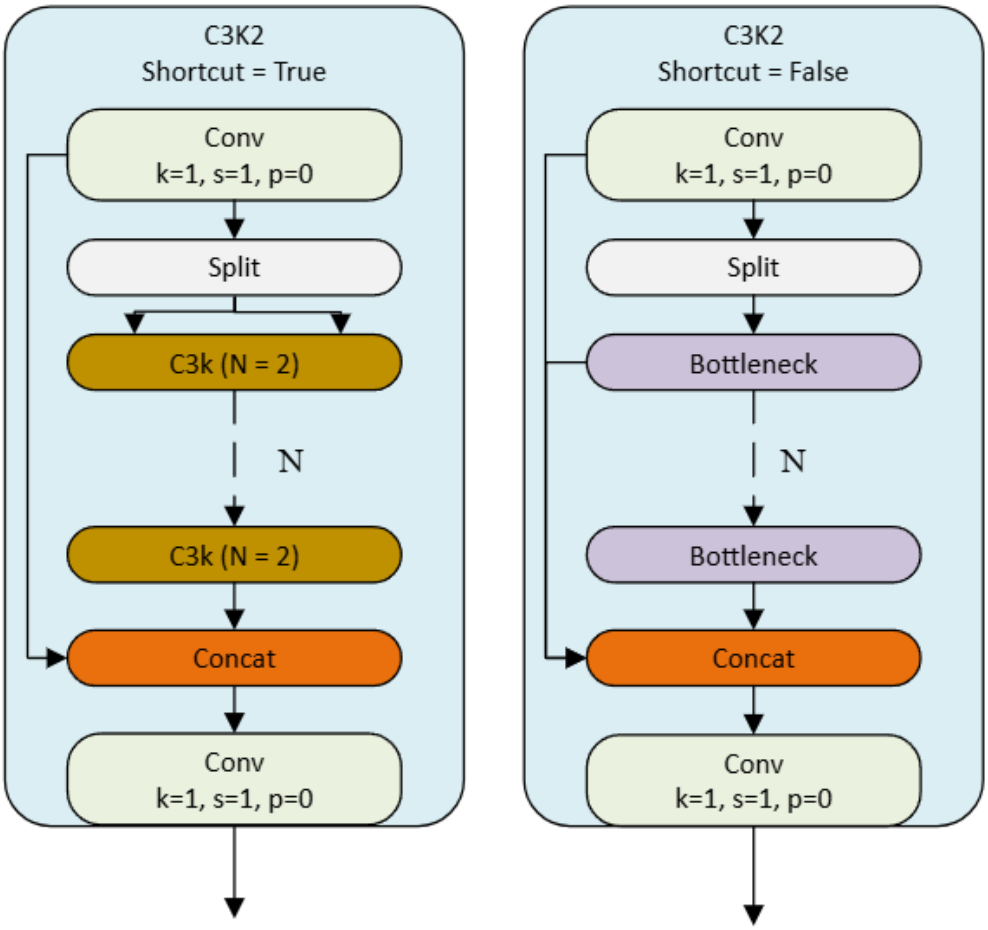
YOLOv11, önceki sürümlerle benzer bir mimariyi takip ederek, görüntüyü indirgemek için başlangıçta evrişim katmanlarını kullanmaktadır. Bu katmanlar, uzaysal boyutları kademeli olarak küçültürken kanal sayısını artırarak özellik çıkarma sürecinin temelini oluşturmaktadır. Daha derin katmanlara geçtikçe model, daha soyut ve yüksek seviyeli özellikleri öğrenebilir hale gelmektedir [22]. YOLOv11, kendisinden önceki versiyonlarda da rastlayabileceğimiz üzere evrişim katmanlarında iki boyutlu evrişim bloğunun çıkışına bağlı iki boyutlu Batch Normalizasyonu, aktivasyon fonksiyonu olarak ise Sigmoid Weighted Linear Unit (SiLU) aktivasyon fonksiyonunu kullanmaktadır. Şekil 3.18.'de modelde de "Conv" bloğu olarak adlandırılan bir evrişim katmanının iç içe geçmiş yapısı gösterilmiştir.



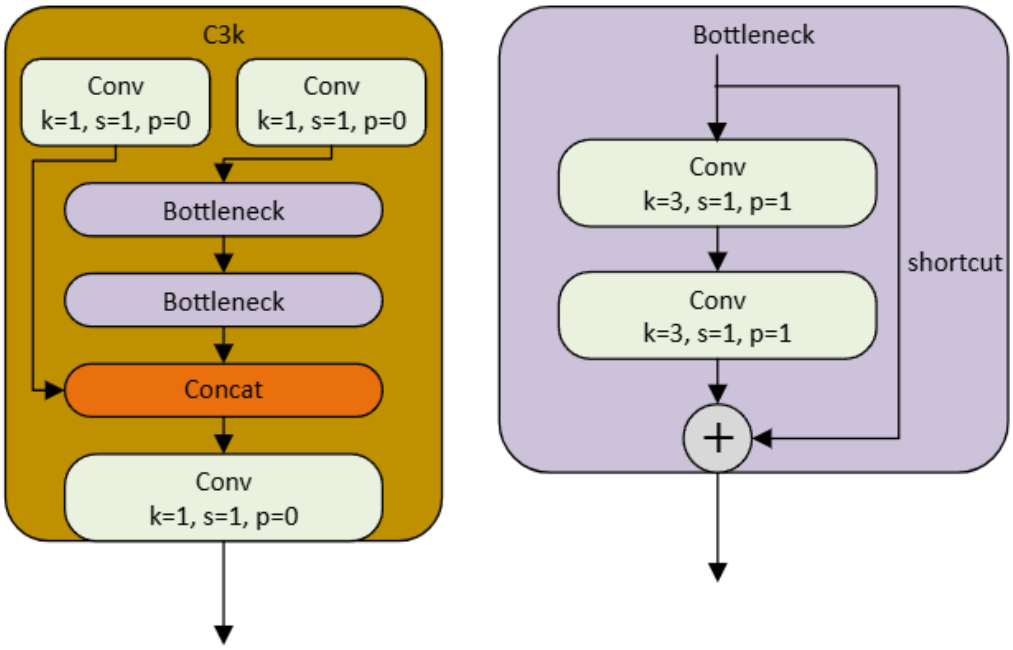

**Şekil 3.18.** Evrişim Katmanı

### 3.6.1.2. C3k2 Bloğu

YOLOv11'de yapılan en önemli iyileştirmelerden biri, önceki versiyonlarda kullanılan C2f bloğunun yerine C3k2 bloğunun getirilmesidir. C3k2, Cross Stage Partial (CSP) Darboğaz (Bottleneck) yapısının daha verimli bir uygulaması olup, büyük tek bir evrişim yerine iki daha küçük evrişim kullanarak hesaplama maliyetini düşürür. Bu değişiklik, modelin hem hızını artırır hem de donanım gereksinimlerini optimize eder. "k2" ifadesi, daha küçük bir çekirdek boyutunu ifade eder ve bu sayede hız kazanımı sağlanırken modelin performansı korunur. Yapılan optimizasyonlar sayesinde YOLOv11, nesne tespitinde daha hızlı ve verimli bir yaklaşım sunarak gerçek zamanlı uygulamalar için daha uygun hale gelmiştir [22]. Model içerisinde C3k2 bloklarının iki türü bulunmaktadır. Bu iki tür, kısayol (shortcut) ifadesinin doğru ya da yanlış (True/False) olarak belirtilmesi ile belirlenmektedir ve kısayolun doğru olduğu C3k2 bloklarında C3k bloğu yer alırken, yanlış olarak belirtilen C3k2 bloklarında sadece darboğaz blokları bulunmaktadır. C3K2, bilgiyi işlemek için C3k bloğunu kullanır. Başlangıçta ve sonda yer alan iki evrişim bloğuna sahiptir. Bu bloklar arasında bir dizi C3K bloğu bulunur. Son aşamada, evrişim bloğunun çıktısı ile son C3k bloğunun çıktısı birleştirilir ve işlem son bir evrişim bloğuyla tamamlanır. Bu yapı, CSP mimarisinden yararlanarak hız ve doğruluk arasında bir denge kurmayı hedefler. C3k2 ve C3k bloklarında yer alan "N" sayısı ise YOLOv11'in varyantlarına göre değişiklik göstermektedir. YOLOv11n varyantında ise N sayısı 1'e eşittir. Şekil 3.19.'da C3k2 bloğunun doğru ve yanlış kısayol ifadeleri ile ayrılan iki türünün modeli, Şekil 3.20.'de C3k bloğunun ve Darboğaz bloğunun modeli verilmiştir.




**Şekil 3.19.** C3k2 Bloğu



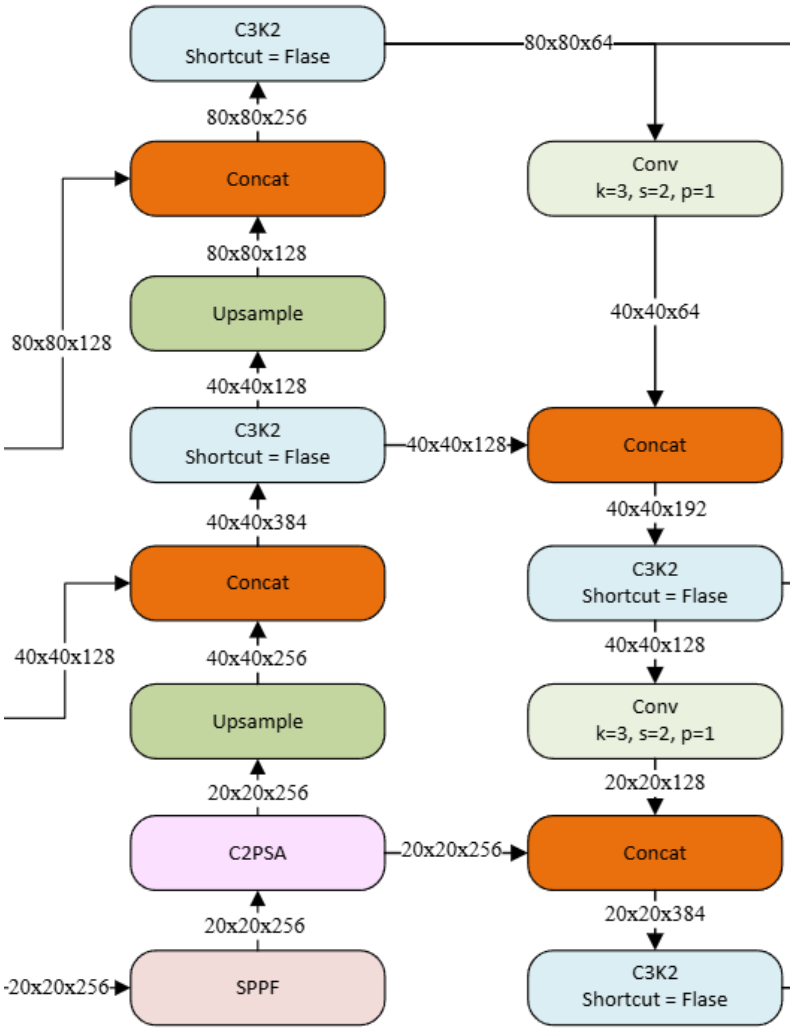

**Şekil 3.20.** C3k ve Bottleneck Blokları

### 3.6.2. Boyun (Neck)

Boyun katmanı, omurga tarafından çıkarılan özellik haritalarını daha anlamlı hale getiren bir ara katmandır. Bu katmanda bir dizi ağ katmanı kullanılarak farklı ölçeklerdeki özellikler birleştirilir ve daha zengin bilgi üretilir. Özellikle, FPN (Feature Pyramid Network) ve PANet (Path Aggregation Network) gibi yapılar boyun katmanında sıkça kullanılır. Bu katman sayesinde model, farklı boyutlardaki nesneleri daha iyi algılayabilir ve hem küçük hem de büyük nesneler için daha iyi tespit performansı sağlar. Boyun katmanının temel görevleri şunlardır; özellik haritalarını birleştirerek daha güçlü bir temsil oluşturmak, küçük nesnelerin tespitini iyileştirmek ve konum bilgilerini daha hassas hale getirmek [86].

Boyun (Neck), farklı ölçeklerdeki özellikleri birleştirir ve tahmin için başa (Head) iletir. Bu süreç genellikle özellik haritalarının farklı seviyelerden yükseltilmesi (upsampling) ve birleştirilmesini (concatenation) içerir, bu sayede model çoklu ölçek bilgilerini etkili bir şekilde yakalayabilmektedir [22].

YOLOv11, farklı ölçeklerdeki görüntü bölgelerinden özellikleri havuzlamak için tasarlanmış olan SPPF (Spatial Pyramid Pooling Fast) modülünü korumaktadır. Bu modül, modelin farklı boyutlardaki nesneleri, özellikle küçük nesneleri daha iyi yakalamasını sağlamaktadır. Küçük nesnelerin tespiti, önceki YOLO sürümlerinde yüksek bir zorluk seviyesinde bir problem teşkil etmekteydi ve SPPF, bu sorunu çözmeye yönelik önemli bir iyileştirme sunmaktadır [87]. Şekil 3.21.'de YOLOv11'in boyun (neck) yapısı gösterilmektedir.



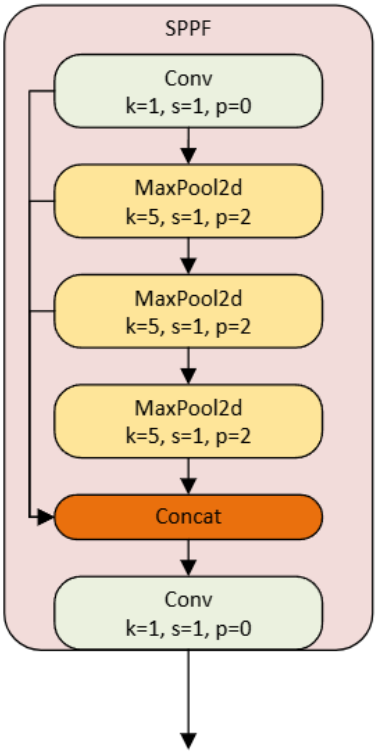

**Şekil 3.21.** YOLOv11 Boyun (Neck) Yapısı

### 3.6.2.1. SPPF ve C2PSA

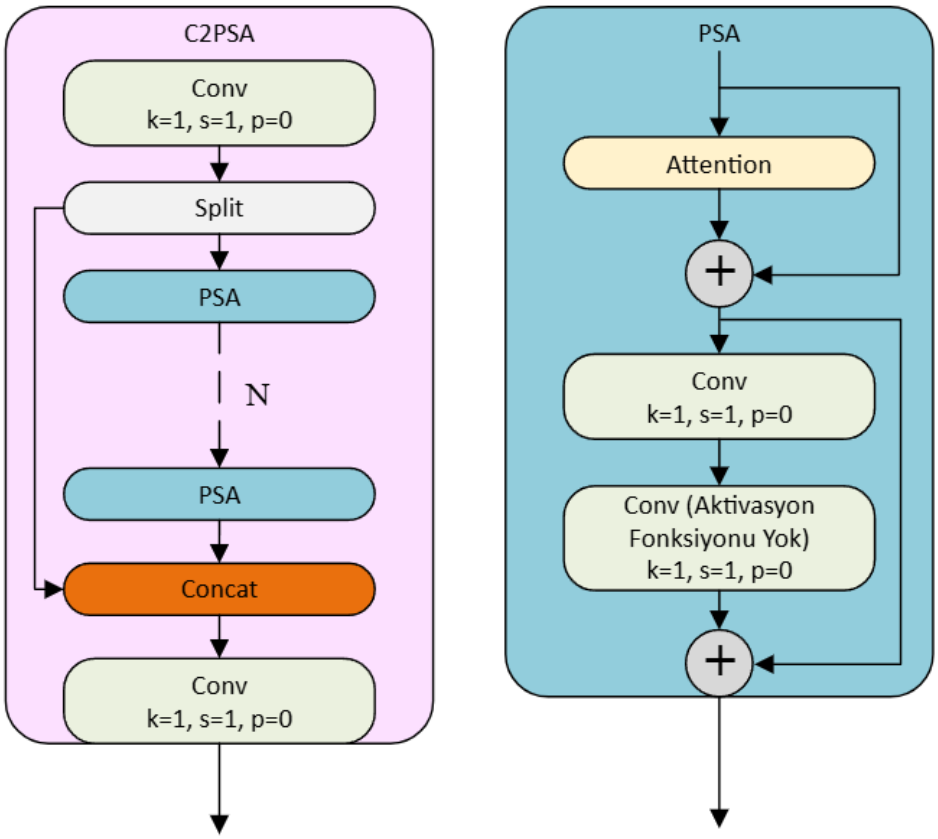
YOLOv11, önceki sürümlerden Spatial Pyramid Pooling Fast (SPPF) bloğunu korurken, bunun sonrasında yeni bir Cross Stage Partial with Spatial Attention (C2PSA) bloğu eklemiştir. C2PSA bloğu, özellik haritalarındaki uzaysal dikkati artırarak modelin görsel bilgiye odaklanma biçimini geliştirmiştir. Bu uzaysal dikkat mekanizması, modelin önemli alanlara daha derinlemesine odaklanmasını sağlayarak, tespit performansını iyileştirmektedir. C2PSA bloğu, özellikle görüntüdeki farklı boyutlardaki nesneleri doğru şekilde tanımlama yeteneğini geliştirmektedir. Bu mekanizma sayesinde model, görüntüdeki belirli alanları daha hassas bir şekilde analiz edebilir, bu da farklı boyutlardaki ve farklı pozisyonlardaki nesnelerin tespit doğruluğunu artırma olasılığına sahiptir [22].

SPPF, özellikleri birden fazla maksimum havuzlama (max-pooling) havuzlayarak çok ölçekli bağlamsal bilgileri birleştirmektedir. Bu modül, modelin küçük nesneleri tanıyabilmesini sağlar çünkü farklı çözünürlüklerden gelen bilgileri etkili bir şekilde bir araya getirmektedir. SPPF bloğunun dahil edilmesi, YOLOv11'in gerçek zamanlı hızını korurken, aynı zamanda farklı ölçeklerdeki nesneleri tespit etme yeteneğini geliştirmesini sağlamaktadır. Bu mekanizma, modelin hem hızını hem de doğruluğunu artırarak çoklu çözünürlüklerdeki nesneleri daha etkili şekilde tanımasına olanak tanımaktadır [87].

C2PSA bloğu, PSA (Partial Spatial Attention) modüllerini kullanarak, her biri özellik haritasının ayrı dallarında çalışan iki modülden oluşmaktadır ve sonunda bu modüller birleştirilmektedir. Bu yapı, C2F bloğu yapısına benzer bir şekilde işlemektedir. PSA modüllerinin kullanımı sayesinde, modelin uzaysal bilgilere odaklanmasını sağlanırken, aynı zamanda hesaplama maliyeti ile tespit doğruluğu arasında dengeli bir ilişki kurulmaktadır. C2PSA bloğu, çıkarılan özellikler üzerine uygulanan uzaysal dikkat mekanizması sayesinde modelin, önemli bölgeleri daha hassas bir şekilde seçmesine olanak tanımaktadır. Daha hassas seçimin sonucunda ise YOLOv11'in nesne tespiti konusunda daha ince detayları yakalayabilmesini sağlanabilmektedir. Bu özellik, YOLOv11'i daha önceki sürümlere, özellikle de YOLOv8'e kıyasla, detaylı nesne tespitinin kritik olduğu senaryolarda daha başarılı hale getirmektedir. Model, bu sayede yüksek doğrulukla daha küçük ve karmaşık nesneleri dahi doğru şekilde tespit edebilmektedir [87]. Şekil 3.22.'de SPPF bloğunun, Şekil 3.23.'te ise C2PSA ve PSA bloklarının yapıları verilmiştir.



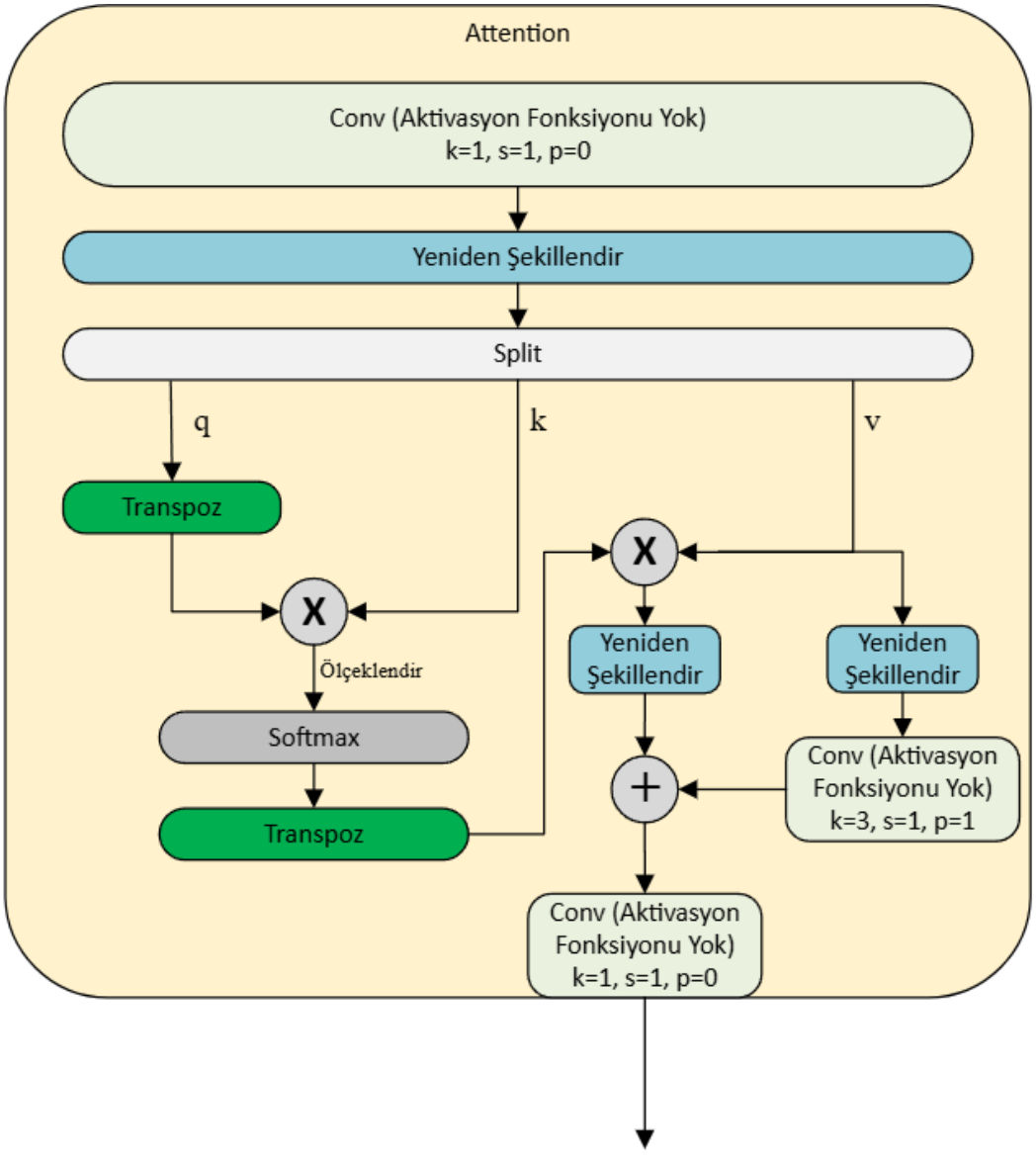

**Şekil 3.22.** SPPF Bloğu




**Şekil 3.23.** C2PSA ve PSA Blokları

### 3.6.2.2. Dikkat Mekanizması (Attention Mechanizm)

YOLOv11'e dikkat çeken yeniliklerden bir tanesi olan C2PSA modülü, kendisi aracılığıyla artan uzamsal dikkat odaklanması sağlamaktadır. Bu dikkat mekanizması, modelin görseldeki önemli bölgelere odaklanmasına olanak tanımakta ve özellikle küçük ya da kısmen gizlenmiş nesnelerin daha doğru bir şekilde tespit edilmesine yardımcı olabilmektedir. C2PSA'nın eklenmesi, YOLOv11'in eski versiyonlarından bir tanesi olan YOLOv8'den ayıran bir özellik olarak öne çıkmaktadır, çünkü YOLOv8'de bu özel dikkat mekanizması bulunmamaktadır. Dikkat mekanizmasında bahsi geçen transpoz operasyonları (-2,-1) oranında bir transpoz işlemidir. Yani matrisin transpozu alındıktan sonrasında ayna görüntüsü de alınmalıdır. YOLOv11'in sahip olduğu dikkat mekanizmasının modeli Şekil 3.24.'te adım adım gösterilmiştir.

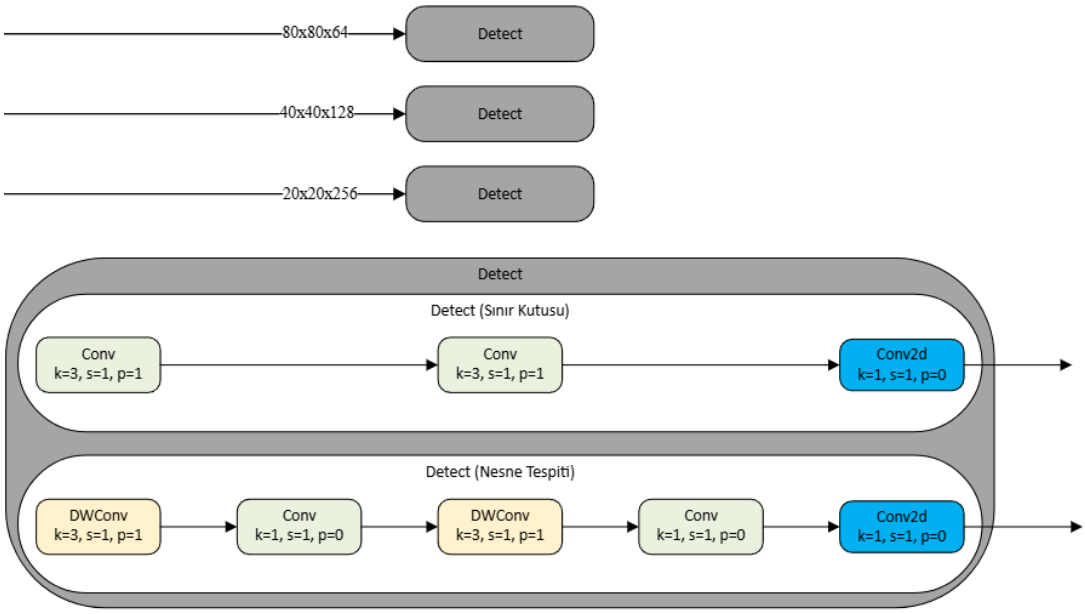



**Şekil 3.24.** Dikkat Mekanizması Modeli

### 3.6.3. Kafa (Head)

YOLOv11 mimarisinde kafa (head) bölümü, modelin son aşamasını oluşturur ve işlenmiş özellik haritalarını (feature maps) doğrudan nesne tespiti sonuçlarına dönüştürür. Bu bölümde yer alan "Detect" modülü, Kafanın en temel bileşenidir ve çoklu ölçeklerdeki özellik haritalarını işler. Bu işlem, bir görüntüye hem yakınlaşıp hem de uzaklaşıyormuş gibi farklı seviyelerden bakmayı sağlar. Böylece hem büyük nesneler (örneğin makineler) hem de küçük nesneler (örneğin aletler) hassas bir şekilde tespit edilebilir. Bu çok ölçekli algılama yeteneği, farklı boyutlardaki nesneleri güvenilir biçimde tanımak için oldukça kritiktir [88, 89]. Baş katmanının iki temel çıktısı vardır. Bunlardan ilki nesne sınıflandırmadır. Model, belirlenen özelliklere dayanarak görüntüde tespit edilen nesnelerin hangi sınıfa ait olduğunu belirler. Örneğin, bir görüntüde "araba", "insan" veya "kedi" gibi nesnelerin varlığını tahmin edebilir. İkinci temek çıktı ise sınır kutusu koordinatlarıdır. Model, tespit edilen nesnelerin konumlarını belirlemek için sınırlayıcı kutuların (bounding box) kesin koordinatlarını üretir. Bu sayede nesnelerin görüntüde nerede bulunduğu belirlenir [22].

Önceki YOLO sürümlerinde olduğu gibi, YOLOv11 de farklı boyutlardaki nesneleri tespit edebilmek için çok ölçekli (multi-scale) bir tahmin başlığı kullanır. Bu kafa, omurga ve boyun tarafından üretilen özellik haritalarını kullanarak düşük, orta ve yüksek olmak üzere üç farklı ölçekte tahmin kutuları üretir [87]. YOLOv11'in kendinden önceki versiyonların genelinden farklılık gösteren bir diğer kısmı ise kafa kısmında bulunan evrişim katmanlarındadır. Diğer versiyonlarda normal evrişim katmanları bulunurken YOLOv11'de nesne tespit alanında evrişim katmanları arasında Derinlik Boyunca Evrişim (Depthwise Convolution – DWConv) katmanları bulunmaktadır. DWConv ile Conv arasındaki fark ise DWConv'da her matris kanalı için aynı filtre uygulanmaktadır. YOLOv11'in kafa (head) yapısı ve bir tespit (detect) bloğunun iç yapısı Şekil 3.25.'te görülebilmektedir.




**Şekil 3.25.** YOLOv11 Kafa (Head) Yapısı ve Tespit (Detect) Bloğu

## 3.7. Performans Değerlendirme Ölçütleri

Makine öğrenimi modellerinin performansını analiz etmek için sıkça kullanılan araçlardan biri karmaşıklık matrisidir (confusion matrix). Bu matris, modelin yaptığı tahminlerle gerçek sınıflandırmaları yan yana koyarak doğruluk ve hata oranlarını detaylı bir şekilde inceleme imkanı sunmaktadır. Tablo 3.3.'te gösterilen bu yapı, hangi sınıfların doğru tahmin edildiğini, hangi sınıfların karıştırıldığını ve modelin güçlü ya da zayıf yönlerini belirlemeye yardımcı olan kritik bilgileri içermektedir. Özellikle yanlış sınıflandırmaların nerede yoğunlaştığını gözlemleyerek modelin geliştirilmesi gereken alanlarını belirlemek için önemli bir referans noktasıdır [90].

**Tablo 3.3.** Karmaşıklık Matrisi Değerlendirmesi

| | | **Tahmin Değeri** | |
|---|---|---|---|
| | | POZİTİF | NEGATİF |
| **Asıl Değer** | POZİTİF | Doğru Pozitif (DP)<br>(True Positive - TP) | Yanlış Pozitif (YP)<br>(False Positive - FP) |
| | NEGATİF | Yanlış Negatif (YN)<br>(False Negative - FN) | Doğru Negatif (DN)<br>(True Negative - TN) |

Duyarlılık (Sensitivity), bir modelin pozitif olan örnekleri doğru bir biçimde pozitif olarak sınıflandırma konusundaki başarımını ifade etmektedir. Bir başka deyişle, pozitif

sınıfa ait verilerin ne kadarının model tarafından doğru şekilde yakalanabildiğinin bir göstergesidir. Matematiksel olarak Eşitlik 3.7'te bulunan denklemle hesaplanmaktadır.

$$Duyarlılık = \frac{DP}{DP+YP} \tag{3.7}$$

Özgüllük (Spesifite), modelin negatif olan örnekleri hatasız biçimde negatif olarak tanıma yeteneğini yansıtan bir başarı ölçütüdür. Matematiksel denklemi Eşitlik 3.8'de bulunmaktadır.

$$\text{Özg}\ddot{\text{u}}\text{ll}\ddot{\text{u}}k = \frac{DN}{DN+YN} \tag{3.8}$$

Doğruluk (Accuracy), modelin tüm sınıflara ait örnekleri doğru bir şekilde tahmin etme konusundaki genel performansını yansıtan temel bir değerlendirme ölçütüdür. Eşitlik 3.9'daki denklem doğruluk parametresini hesaplamanın yolunu göstermektedir.

$$Doğruluk = \frac{DP+DN}{DP+DN+YP+YN} \tag{3.9}$$

Kesinlik (Precision), modelin pozitif olarak sınıflandırdığı örneklerin ne kadarının gerçekten pozitif olduğunu ortaya koyan isabet oranıdır. Eşitlik 3.10'da kesinlik parametresinin hesaplanması için gereken denklem verilmiştir.

$$Kesinlik = \frac{DP}{DP+YP} \tag{3.10}$$

Duyarlılık (Recall), veri setindeki tüm pozitif örneklerden modelin doğru bir şekilde tespit ettiği pozitif tahminlerin oranını ifade etmektedir. Başka bir deyişle, modelin gerçekten pozitif olan örnekleri yakalama başarısını göstermektedir. Yüksek duyarlılık, eksik pozitif örneklerin minimumda tutulduğunu belirtmektedir. Matematiksel olarak Eşitlik 3.11'deki denklem ile hesaplanabilmektedir.

$$Duyarlılık = \frac{DP}{DP+YN} \tag{3.11}$$

F-Skor (F-Score), kesinlik ve duyarlılığın birleşimiyle modelin genel performansını ölçen bir metriktir. Bu metrik, modelin hem doğruluğunu hem de eksik tahmin yapma eğilimini dikkate alarak, iki metriğin dengesini yansıtır. Yüksek bir F-Skor, yanlış pozitiflerin ve yanlış negatiflerin düşük olduğu, genel olarak güvenilir bir modeli ifade eder. Eşitlik 3.12'deki denklem ile F-Skor hesabı yapılabilmektedir.

$$F - Skor = \frac{2DP}{2DP+YP+YN} \tag{3.12}$$

Nesne tespiti (object detection) alanında yapılan değerlendirmelerde, "doğru negatif" (True Negative - TN) kavramının anlamlı bir şekilde kullanılabilir olmadığına dikkat çekmek büyük önem taşımaktadır. Bunun temel nedeni, herhangi bir görüntü içerisinde yer almaması gereken, yani modelin tespit etmemesi gereken potansiyel sınırlayıcı kutuların teorik olarak sonsuz sayıda bulunmasıdır. Bu durum, klasik sınıflandırma problemlerindeki negatif örnek tanımıyla doğrudan örtüşmediği için, nesne tespiti değerlendirme metriklerinde DN teriminin pratik bir karşılığı bulunmamaktadır. Dolayısıyla, DN üzerinden yapılan analizler, bu bağlamda sağlıklı ve anlamlı sonuçlar üretmekten uzaktır [91].

Yukarıdaki tanımlar, "doğru tespit" ve "yanlış tespit" kavramlarının ne olduğunun belirlenmesini gerektirir. Bunun yaygın bir yöntemi, kesişim-birleşim oranı (IOU) kullanmaktır. IOU, iki veri kümesinin benzerliğini ölçen bir Jaccard İndeksi'ne dayanan bir değerlendirme yöntemidir [92]. Obje tespiti çerçevesinde IOU, tahmin edilen sınırlayıcı kutular ($SK_t$) ve temel gerçek sınırlayıcı kutuların ($SK_{tg}$) kesiştiği bölgenin alanının bu kutuların birleşiminin alanına oranını ölçer. Eşitlik 3.13'te verilen denklem ile bu hesap yapılabilmektedir.

$$J(SK_t, SK_{tg}) = IOU = \frac{alan(SK_t \cap SK_{tg})}{alan(SK_t \cup SK_{tg})} \tag{3.13}$$

Daha önceden de bahsedildiği üzere nesne tespitinde DN'lerin bir karşılığı bulunmadığından dolayı nesne tespitinde baz alınabilecek iki kavram kesinlik ve duyarlılık kavramlarıdır. Kesinlik × duyarlılık eğrisi, tespit cihazının oluşturduğu sınırlayıcı kutular için farklı güven seviyelerinde kesinlik ve duyarlılık arasındaki dengeyi gösterir. Eğer tespit cihazının güven seviyesi öyle bir yerdeyse ki yanlış pozitif (YP) sayısı düşükse, kesinlik yüksek olur. Ne yazık ki bu eğri bazı pratik uygulamalarda bir zikzak eğrisine dönüşebilir.

Bu yüzden ortalama kesinlik (Average Precision - AP) hesabı yapılarak eğrinin zikzak hali alması önlenebilir. Bunu yapmanın iki yolu vardır. Bunlardan ilki Eşitlik 3.14'te denklemi verilmiş olan, kesinlik ($P$) × duyarlılık ($R$) eğrisinin şeklini 11 eşit aralıklarla belirlenmiş duyarlılık seviyelerinde maksimum kesinlik değerlerinin ortalaması alınarak özetlenen 11-noktalı interpolasyondur. Bir diğeri ise denklemi Eşitlik 3.15'te verilmiş olan, eşit aralıklı 11 noktadan oluşan bir interpolasyon yerine, tüm noktalardan oluşan bir interpolasyon yapılan tüm noktalı interpolasyon yöntemidir.

$$AP_{11} = \frac{1}{11} \sum_{R \in \{0, 0.1, \ldots, 0.9, 1\}} P_{interp}(R)$$

ve (3.14)

$$P_{interp}(R) = \max_{\tilde{R}: \tilde{R} \geq R} P(\tilde{R})$$

Bu AP tanımında, her bir duyarlılık seviyesi $R$'de gözlemlenen kesinlik $P(R)$ yerine, duyarlılık değeri $R$'den büyük olan maksimum kesinlik $P_{interp}(R)$ dikkate alınarak AP hesaplanmaktadır.

$$AP_{all} = \sum_{n} (R_{n+1} - R_n) P_{interp}(R_{n+1})$$

ve (3.15)

$$P_{interp}(R_{n+1}) = \max_{\tilde{R}: \tilde{R} \geq R_{n+1}} P(\tilde{R})$$

Bu durumda ise, yalnızca birkaç noktadaki gözlemlenen kesinlik yerine, her seviyedeki kesinliğin enterpolasyonu ile elde edilir; burada, duyarlılık değeri $R_{n+1}$'den büyük veya ona eşit olan maksimum kesinlik dikkate alınır.

Ortalama AP (mAP) ise, nesne tespit cihazlarının belirli bir veri tabanındaki tüm sınıflar üzerindeki doğruluğunu ölçmek için kullanılan bir metriktir. mAP, tüm sınıflar üzerindeki AP değerlerinin basitçe ortalamasıdır. Eşitlik 3.16'da verilen denklem ile hesaplanabilir [91].

$$mAP = \frac{1}{N} \sum_{i=1}^{N} AP_i \quad (3.16)$$

AP değerinin farklı varyasyonları bulunmaktadır. AP, farklı IOU değerleriyle değerlendirilmektedir. Genellikle %50 ile %95 arasında 5'erlik adımlarla değişen 10 farklı IOU için hesaplanmakta ve genellikle AP@50:5:95 olarak raporlanmaktadır. Ayrıca, IOU'nun tekil değerleriyle de değerlendirilebilir; en yaygın kullanılan değerler %50 ve %75 olup sırasıyla AP50 ve AP75 olarak raporlanmaktadır. Bu AP değerleriyle hesaplanan ortalama AP değerleri de mAP50-95, mAP50 ve mAP75 şeklinde raporlanmaktadır [91].

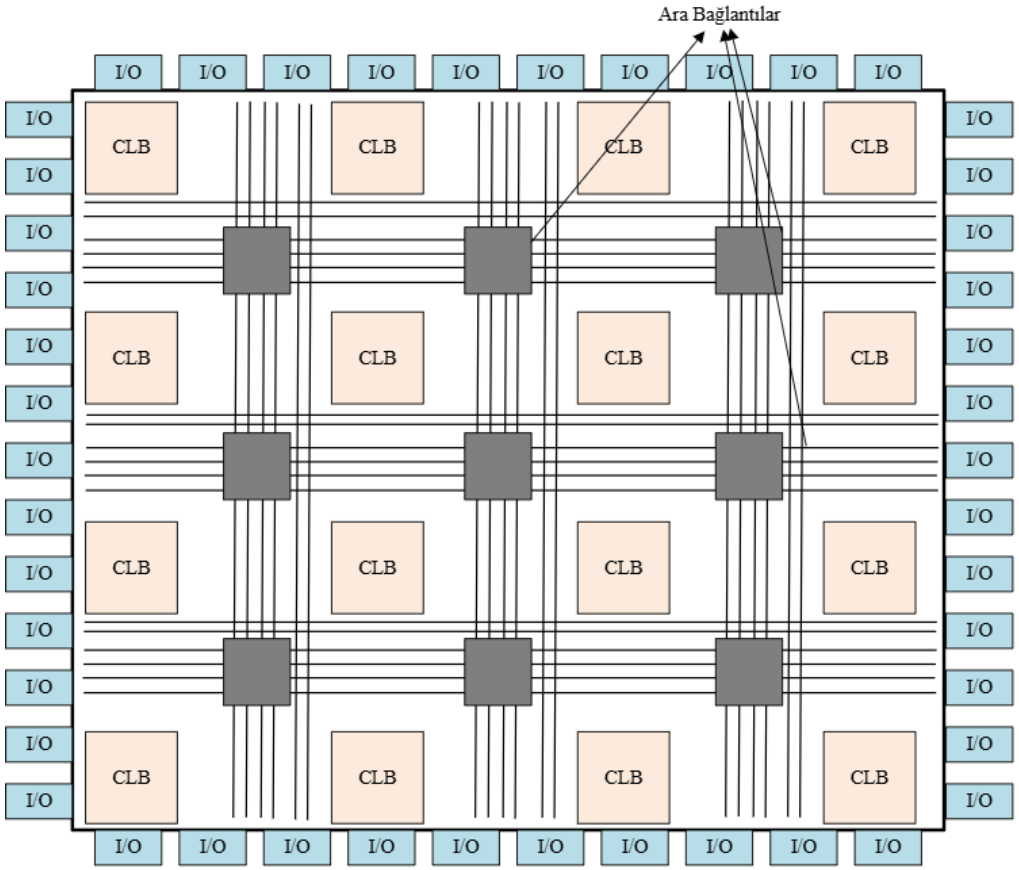
Bu tez çalışmasında kullanılan değerlendirme metrikleri ise kesinlik (P), duyarlılık (R), mAP50 ve mAP50-95 değerlendirme ölçütleridir.

# 4. ALANSAL PROGRAMLANABİLİR KAPI DİZİSİ (FPGA)

Programlanabilir devrelerin geçmişi, 1970'li yıllarda programlanabilir mantık dizilerinin üretilmesine dayanmaktadır [93]. FPGA (Field Programmable Gate Array), 1984 yılında AMD Xilinx firması tarafından geliştirilen bir teknolojidir ve Türkçe karşılığı Alanında Programlanabilir Kapı Dizisi'dir. FPGA modeli, çip tasarım süreçlerine yeni bir perspektif kazandırmıştır. ASIC gibi sabit çip tasarımlarındaki basit değişiklikler, genellikle tüm yapının bozulmasına ve çipin yeniden tasarlanmasına neden olurken, FPGA ile bu tür değişiklikler, çipin yeniden yapılmasını gerektirmeden uygulanabilir bir hale getirilmiştir. FPGA, silikon teknolojisiyle üretilmiş ve milyonlarca lojik kapı barındıran, programlanabilir bir yapıya sahiptir, bu da kullanıcılara tasarım üzerinde esneklik ve yeniden programlama imkanı sunmaktadır [23]. Bazı üretim firmaları tarafından "do-it-yourself processor" adıyla anılarak çipin tamamen kullanıcı kontrolünde ve bir yapboz edasıyla kullanılabileceği ifade edilen bu ürün sayesinde eğitim ve çalışma dünyasında birçok insan kendi işlemcilerini eskisinden daha kolay bir şekilde oluşturma imkanı bulmuşlardır. FPGA'ların piyasadaki popülerliklerinin ana sebeplerinden bir tanesi, daha önce geliştirilmiş olan PLD cihazlarından hız ve kapasite bakımından çok daha yüksek performans göstermeleridir [93]. FPGA'lar bahsedilen bu hızı kendilerinin en ayırt edici özelliği olan paralel işlem yapma kabiliyetinden, kapasiteyi ise tamamen kullanıcı inisiyatifine dayalı olarak çip içerisinde bulunan hafıza birimlerinden almaktadır. FPGA, bu özelliklerinden dolayı yapay zeka, görüntü işleme, sinyal işleme, havacılık ve uzay gibi hız ve özelleştirilebilir projeler gerektiren alanlarda oldukça yaygın bir şekilde tercih edilmektedir [94, 95].

Bir FPGA teknolojisi mimarisinde programlanabilir bir durumda olan 3 ana bileşen bulunmaktadır. Bu bileşenlerden ilki Konfigüre Edilebilir Mantık Blokları (Configurable Logic Blocks – CLB) olarak bilinmektedir. Bu bloklar FPGA kullanıcısı tarafından oluşturulması hedeflenen lojik devre için fonksiyonel bileşenleri sağlamaktadır. CLB'lerin içerisinde FPGA'nın da temeli olan mantık hücreleri bulunmaktadır. Mantık hücrelerinin yapısında ise Look Up Tablosu (LUT), tek bitlik hafıza birimleri olan Flip-Floplar ve birden fazla seçim yaparken kullanılabilen Çoklayıcılar (Multiplexer) bulunmaktadır. İkinci ana bileşen ise Giriş-Çıkış Blokları (I/O Blocks – IOB) olarak adlandırılmaktadır. IOB'ler programlanabilir giriş/çıkış noktalarıdır. Bu noktalarda bulunan pinler kullanıcının isteği üzerine giriş, çıkış veya hem giriş hem de çıkış olacak şekilde ayarlanabilmektedir.

Giriş/çıkış pinlerinin yönü ayarlanabildiği gibi voltajları da kullanıldığı ortama göre ve FPGA'nın desteklediği voltaj seviyelerine göre farklı voltaj seviyelerinde ayarlanabilmektedir. Üçüncü ve son ana bileşenimiz olan Ara Bağlantılar (Interconnections), IOB ile CLB'yi, IOB ile IOB'yi veya CLB ile CLB'yi birbirine bağlayan ve kullanıcının yazmış olduğu programa göre bloklar arasında bir patika oluşturan bağlantılardır [23, 94, 95]. Şekil 4.1.'de CLB, IOB ve Ara Bağlantılar bir arada gösterilmektedir.




**Şekil 4.1.** CLB, Giriş/Çıkış Blokları ve Ara Bağlantılar

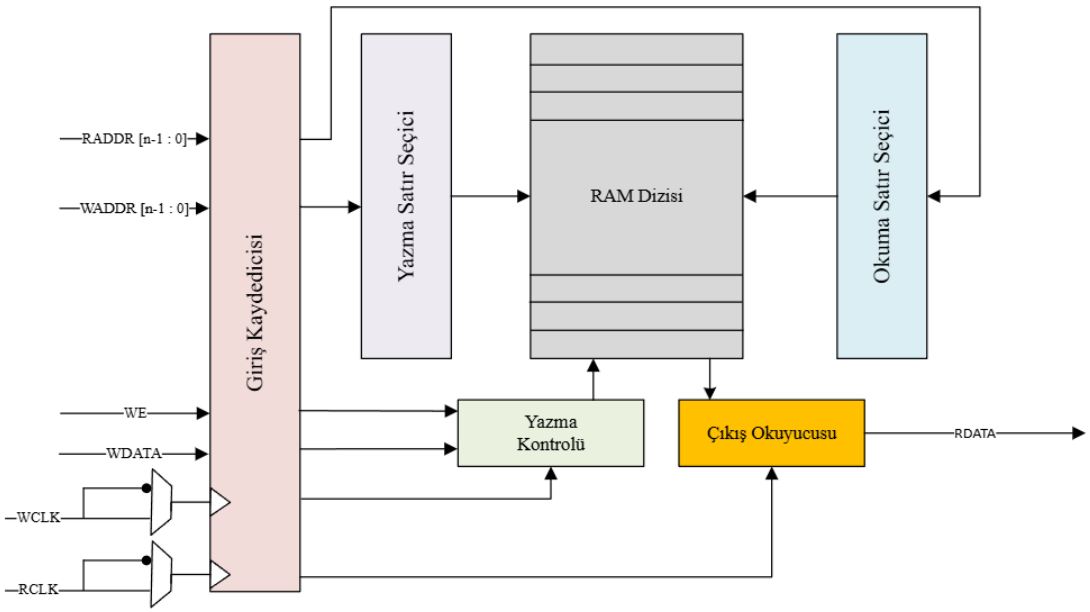
FPGA çiplerini programlamak için normal bir işlemcinin programlanmasından daha farklı bir dil ailesi kullanılmaktadır. Bu dil ailesi 1987 yılında IEEE tarafından kabul edilmiş olan Donanım Tasarım dil sınıfı olarak kabul edilen HDL (Hardware Description Language) dil ailesidir. Bu dil ailesinde ise en bilindik iki dil VHDL (Very High-Speed Integrated Circuit Hardware Description Language) ve Verilog dilleridir. VHDL ve Verilog dilleri geleneksel programlama dillerinden tamamen farklıdır ve akışı ve işleyişi değil, donanımın yapısını, yani elemanların bağlantısını ve entegre devrenin iç konfigürasyonunu ifade etmektedir. VHDL ve Verilog dillerini High-Level Synthesis (HLS) araçlarını kullanarak

geleneksel programlama dillerini kullanarak oluşturmak da mümkündür. Bu yöntem ile kodlama işlemi daha basit bir hal kazanmaktadır. Ne yazık ki HLS yöntemi FPGA içerisindeki zamanlama yönetimi ve kaynak tüketiminin optimizasyonu gerektiren noktalarda etkili değildir [95, 96].

FPGA'nın içerisinde Flip-Flop'lar dışında da RAM (Random Access Memory) ve RAM'den türetilen First-In-First-Out (FIFO) gibi hafıza birimleri yer almaktadır. Bu hafıza birimleri el ile yazılan HDL kodlarıyla oluşturulabileceği gibi başka firmalar veya FPGA üreticisi tarafından oluşturulan ve programlama aracına eklenen, modelleme mantığıyla yapılan tasarımın ihtiyaç halinde yeniden kullanabilecek ve kod akışını hafifleten fonksiyonel donanım tasarımları olan ve IP Çekirdeği adı verilen bloklar çağırılarak da kullanabilmektedir. IP Çekirdekleri sadece hafıza birimlerini içermekle kalmayıp aynı zamanda saat üreteçleri, kayan noktalı gösterim blokları, haberleşme blokları gibi birden fazla çeşitte bulunmaktadırlar.

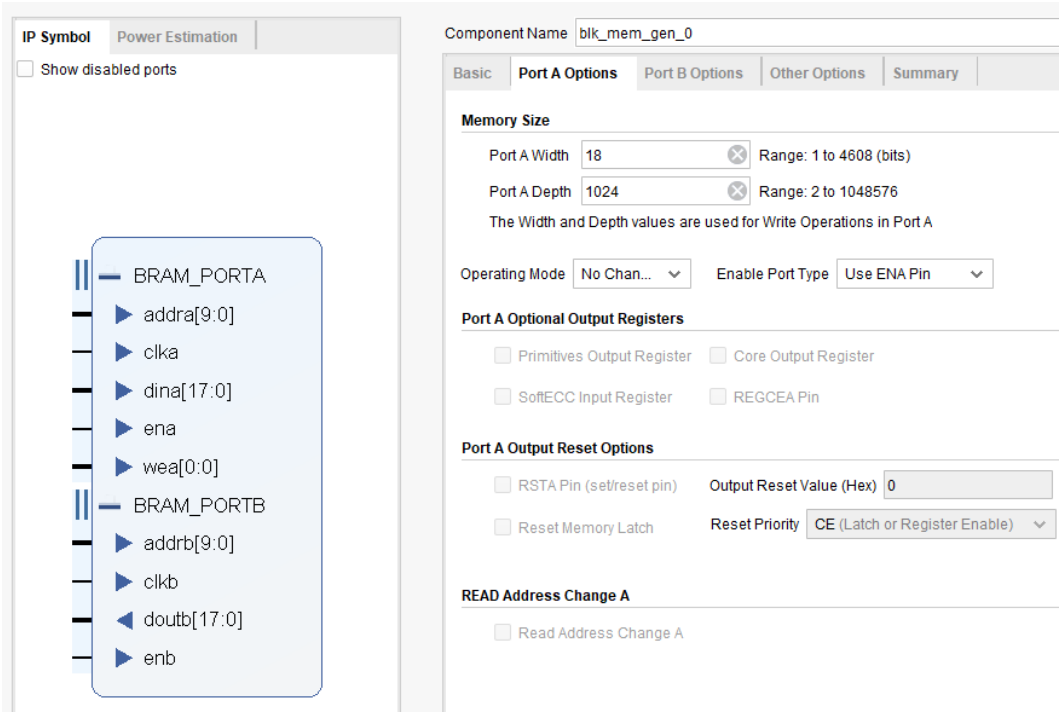
## 4.1. RAM Yapısı

FPGA iç mimarisinde donanımsal olarak yer alan bellek birimleri bulundurmaktadır. Bu bellek birimleri verilerin büyüklüğüne göre iki farklı bellek yapısına sahiptir; daha büyük verilerin saklanabilmesi için bulunan RAM Blokları ve daha küçük verilerin saklanabilmesi adına kullanılan dağınık bellek yapıları bu iki bellek yapısıdır. Dağınık bellek yapısı FPGA'nın ana bileşenlerinden birisi olan mantık hücrelerinin içerisine dağıtılmış bir durumdadır. RAM'ler ise donanımsal olarak bir RAM dizisi şeklinde FPGA mimarisinde yer almaktadır [96]. Blok RAM'ler FPGA içerisinde yazma/okuma işleminin tek bağlı üzerinden yapıldığı tek bağlantı noktalı Blok RAM ve yazma/okuma işlemlerinin farklı bağlantı noktalarından ve aynı anda hem yazma hem de okuma işleminin aynı anda yapılabildiği çift bağlantı noktalı Blok RAM olarak iki ana gruba ayrılabilir. Şekil 4.2.'de örnek bir çift bağlantı noktalı Blok RAM şeması görülebilmektedir.




**Şekil 4.2.** Çift bağlantı noktalı Blok RAM şeması

Kullanıcı tarafından RAM kullanmanın 3 yöntemi bulunmaktadır. Bu yöntemlerden ilki üretici firma tarafından oluşturulmuş IP Çekirdeğini programlama aracında IP katalog aracılığı ile çağırmaktır. İkincisi ise FPGA'nın üretici firmalarına göre değişkenlik gösteren ve FPGA'nın iç yapısında donanımsal olarak var olan ilkelleri (primitive) çağırmaktır. Sonuncusu ise kod içerisinde iki boyutlu bir dizi oluşturarak kullanıcının kendi RAM modülünü yazmasıdır. Kullanıcının kullanmış olduğu RAM boyutları kullanıcının isteğine göre şekillendirilebilir. İlkel kullanarak oluşturulan RAM'ler haricinde diğer yöntemlerde kullanıcı istediği derinlikte ve bit genişliğinde RAM çağırabilmektedir. Programlama aracı sentez esnasında kullanıcının seçmiş olduğu RAM boyutuna göre FPGA'nın içerisindeki hafıza bloklarını rezerve etmektedir. Şekil 4.3.'te AMD Xilinx firmasına ait bir FPGA'da RAM çağırmak için IP katalog ve Şekil 4.4.'te bir Blok RAM ilkel bloğu örneği gösterilmektedir.




**Şekil 4.3.** AMD Xilinx Vivado Çift Bağlantılı Blok RAM IP Kataloğu

```
library IEEE;
use IEEE.NUMERIC_STD.all;
use IEEE.STD_LOGIC_1164.all;
use IEEE.STD_LOGIC_TEXTIO.all;

library STD;
use STD.TEXTIO.all;

library UNISIM;
use UNISIM.VPKG.all;
use UNISIM.VCOMPONENTS.all;

  entity RAMB36E2 is
    generic (
      CASCADE_ORDER_A : string := "NONE";
      CASCADE_ORDER_B : string := "NONE";
      CLOCK_DOMAINS : string := "INDEPENDENT";
      DOA_REG : integer := 1;
      DOB_REG : integer := 1;
      ENADDRENA : string := "FALSE";
      ENADDRENB : string := "FALSE";
      EN_ECC_PIPE : string := "FALSE";
      EN_ECC_READ : string := "FALSE";
      EN_ECC_WRITE : string := "FALSE";
      INITP_00 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
      INITP_01 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
      INITP_02 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
      INITP_03 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
      INITP_04 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
      INITP_05 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
      INITP_06 : std_logic_vector (255 downto 0) := X"0000000000000000000000000000000000000000000000000000000000000000";
```

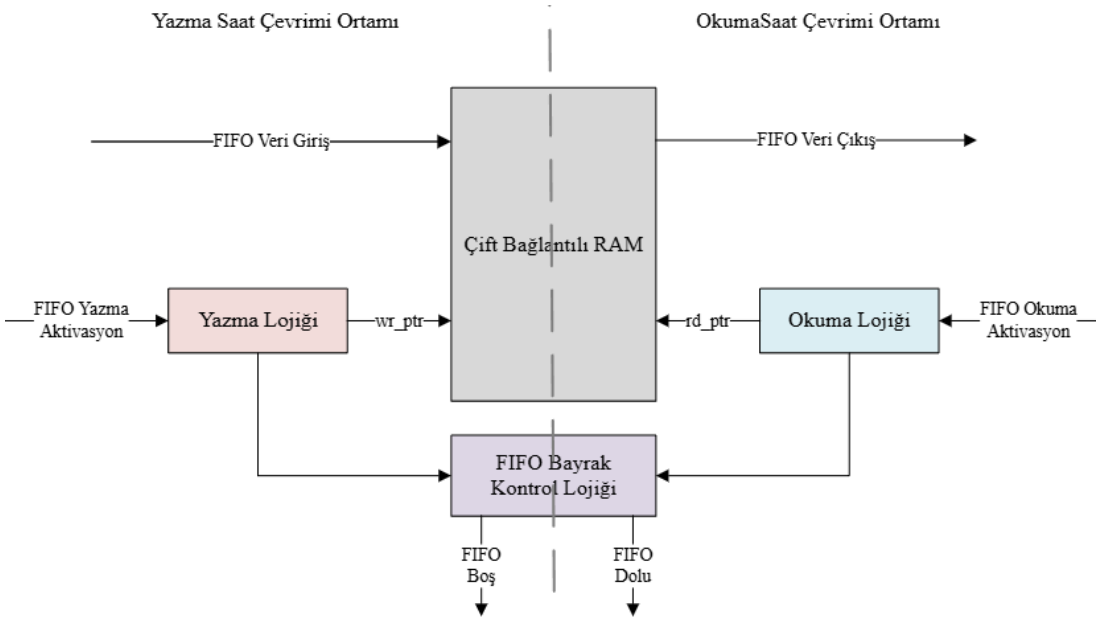
**Şekil 4.4.** 36x2 Çift Bağlantılı Blok RAM İlkel Bloğu Kod Parçası

## 4.2. FIFO Yapısı

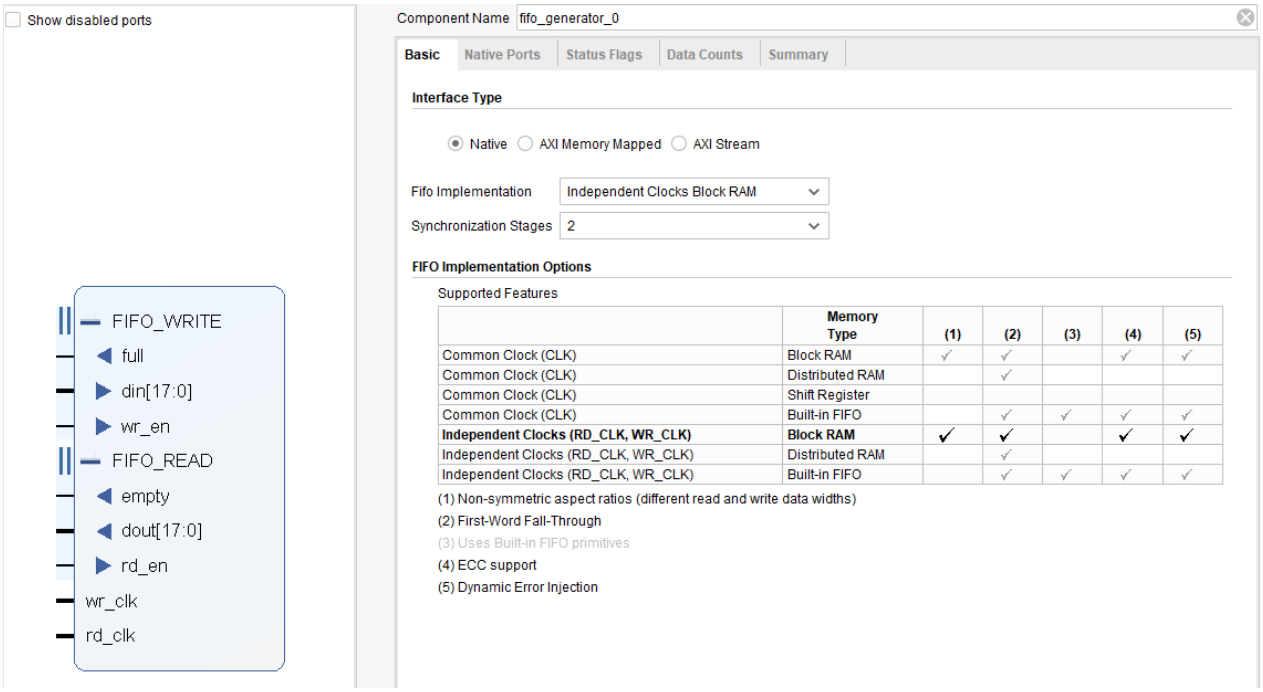
FPGA içerisindeki RAM bloklarını kullanarak uygulama özelinde tanımlanabilen hafıza birimleri oluşturulabilmektedir. Bu hafıza birimlerinden bir tanesi de çoğunlukla elastik bir veri saklama alanı veya iki farklı saat çevriminden gelen veriyi birbirine senkron hale getirmek amacıyla kullanılan İlk Giren İlk Çıkar (First-in-First-Out – FIFO) yapısıdır. FIFO belleği, veri girişi/çıkışı için basit bir protokol kullanılarak adres pinlerinin tamamen

ortadan kaldırılmasıyla ayırt edilmektedir; depolanan veriler, depolanma sırasına göre okunmaktadır [97].

FIFO'nun çalışma mantık modülünün görevi FIFO boş ve dolu bayrak sinyali sağlamaktır, bu sinyal harici devre FIFO'suna kritik bir duruma ulaştığını söyler: eğer dolu sinyal varsa, kritik durum için FIFO yazma işlemi, daha fazla veri depolamak için alan olmadığını; eğer boş sinyal varsa, kritik durum için FIFO okuma işlemi, daha fazla veri FIFO'sunun okunamayacağını belirtir. Bu nedenle boş, FIFO dolu sinyali, engelleyici veri depolamasının kendi bağımsız çalışmasını ve yönetimini elde etmek için onu okumak ve yazmak için çok önemli bir rol oynar [98]. FIFO, aynı zamanda veriyi kullanıcıya sunma açısından ilk kelime düşer (first-word-fall-through – FWFT) özelliğine sahiptir. Bu özelliğin açık olduğu FIFO'larda veri mevcut olduğunda, ilk sözcük FIFO'dan düşer ve otomatik olarak çıkış veri yolunda görünür. Şekil 4.5.'te bir FIFO'nun iç yapısı gösterilmiştir. Şekil 4.6.'da ise AMD Xilinx Vivado platformundaki FIFO IP Kataloğu verilmiştir.




**Şekil 4.5.** FIFO İç Yapısı



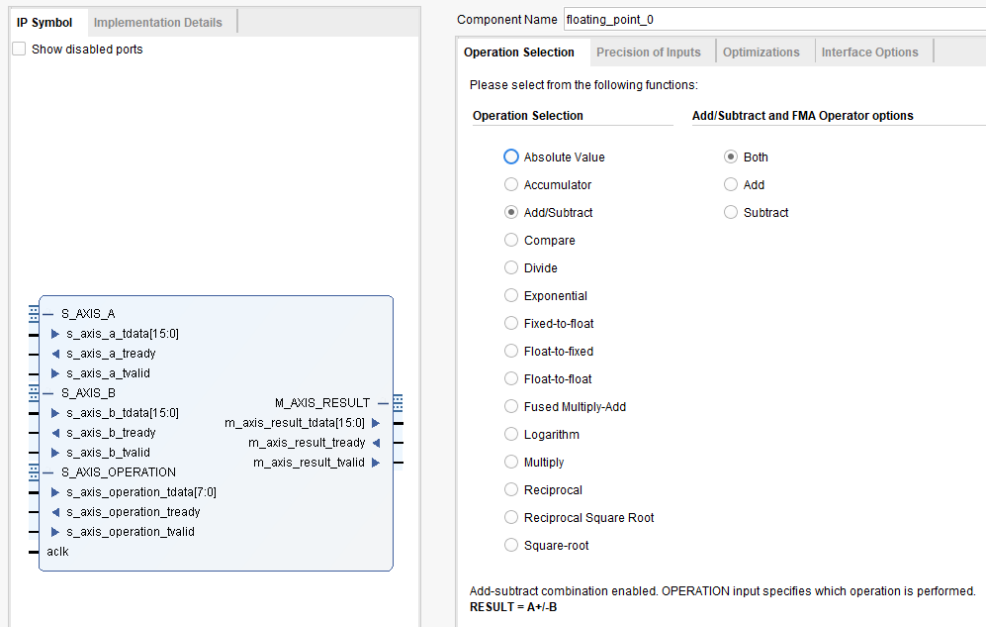

**Şekil 4.6.** AMD Xilinx Vivado FIFO IP Kataloğu

## 4.3. Kayan Noktalı Gösterim IP Çekirdeği

Xilinx Kayan Noktalı Gösterim IP Çekirdeği, FPGA üzerinde bir dizi kayan noktalı gösterim aritmetik işleminin gerçekleştirilmesine olanak tanımaktadır. İşlem, çekirdek oluşturulduğunda belirtilir ve her işlem çeşidinin ortak bir arayüzü bulunmaktadır. Kayan Noktalı Gösterim IP Çekirdeği IEEE-754 Standardında [99] tanımlanandan farklı kesir ve üs kelime uzunluğu aralıklarını desteklemektedir. Bu aralıklardan ilki 11 bitlik kesir ve 5 bitlik üs ile 16 bit ile ondalıklı sayıları ifade eden Yarı Duyarlı Formattır (Half Precision Format). İkincisi ise 24 bitlik kesir ve 8 bitlik üs ile toplam 32 bit ile ondalıklı sayıların ifade edildiği Tek Duyarlı Formattır (Single Precision Format). Sonuncusu ise 53 bitlik kesir ve 11 bitlik üs ile 64 bit kullanılarak kayan noktalı gösterim uygulayan Çift Duyarlı Formattır (Double Precision Format) [100]. Kayan noktalı gösterim ifade edilirken ondalıklı kısmın ifade edildiği bölgeye "Mantissa" adı verilmektedir. Bu tez çalışmasında sayıların kayan noktalı gösterimi olarak Yarı Duyarlı Format kullanılmaktadır.

IP Çekirdek birçok matematiksel operasyon yapabilme kabiliyetine sahiptir; mutlak değer alma, akümülatör, toplama/çıkarma, karşılaştırma, bölme, eksponansiyel üs alma, sabit noktalıdan kayan noktalıya çevirme, kayan noktalıdan sabit noktalıya çevirme, kayan noktalıdan kayan noktalıda çevirme, birleştirilmiş çarpma-toplama, logaritma, çarpma, çarpımsal tersini alma, çarpımsal tersini alıp karekök alma, karekök alma. Çekirdek katalog

üzerinden üretilecek ise her bir operasyon çeşidi için farklı bir çekirdek üretmek gerekmektedir. Şekil 4.7.'de IP katalogda operasyon seçme sekmesi gösterilmiştir.

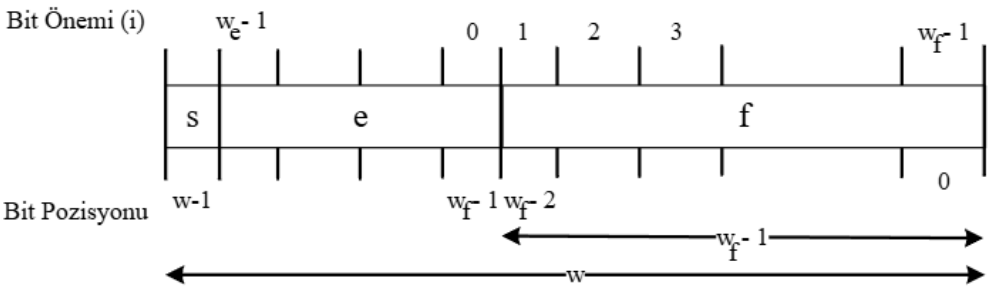



**Şekil 4.7.** Kayan Noktalı Gösterim IP Çekirdeği IP Katalog, Operasyon Seçme

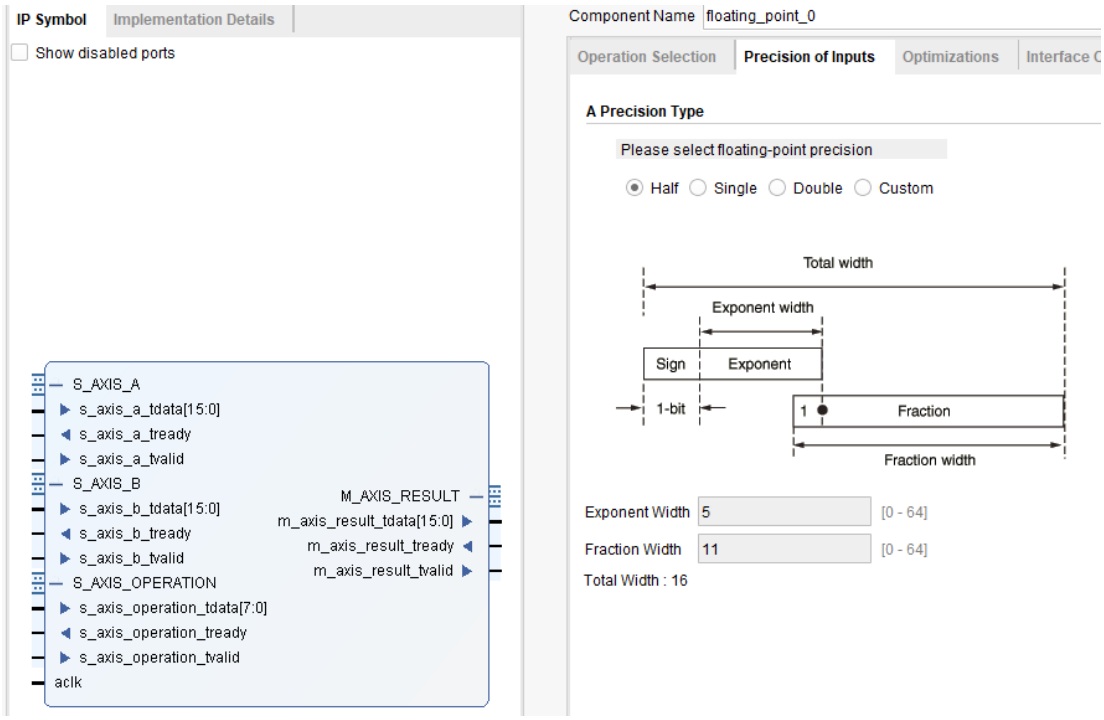
Çekirdek, standart dışı boyutlara izin vermek için IEEE-754 Standardının genelleştirilmiş hali olan kayan noktalı gösterimi kullanmaktadır. Standart boyutlar seçildiğinde, kullanılan biçim ve özel değerler IEEE-754 Standardı tarafından açıklananlarla aynı değerlere sahiptir. Ondalıklı sayılar, işaret, üs ve ondalık (sırasıyla s, e ve $b_0.b_1b_2b_3 \dots b_{w_f-1}$ olarak ifade edilebilir.) kullanılarak gösterilmektedir [100]. Eşitlik 4.1'de ondalıklı sayının işaret, üs ve ondalık cinsinden hesaplanma denklemi verilmiştir.

$$ondalıklı\ sayı = (-1)^s 2^e b_0.b_1b_2b_3 \dots b_{w_f-1} \tag{4.1}$$

Eşitlik 4.1'de $b_i$ olarak verilen sayılar ikilik tabandadır ve katsayısı $2^{-i}$'dir. Bu yüzden $b_0$ sayısı sabit olarak 1 olarak hesaplanmaktadır. Şekil 4.8.'de verildiği üzere bir ondalıklı sayının ikilik tabandaki gösterimi üç ana bölümden oluşmaktadır. Şekil 4.9.'da ise Kayan Noktalı Gösterim IP Çekirdeğinin IP Katalog üzerindeki e ve f parametrelerinin genişliklerinin ayarlanabildiği sekme verilmiştir.




**Şekil 4.8.** Kayan Noktalı Gösterimde Bit Alanları




**Şekil 4.9.** Kayan Noktalı Gösterim IP Çekirdeği IP Katalog - Girdinin Bit Genişliğini Ayarlama Sekmesi

Şekil 4.8.'de "e" parametresiyle verilmiş olan sayı, işaret katsayısı olmadan hesaplanan üs değerini ifade etmektedir. Eşitlik 4.2'de "e" parametresini hesaplamak için gereken denklem verilmiştir. Üs değerinin işaret içeren hali 2'ye tümleyen gösterimi (2's complement representation) ile ifade edilmektedir ve denklemi Eşitlik 4.3'te verilmiştir. Kayan Noktalı Gösterim IP Çekirdeği, sabit noktalı gösterimden kayan noktalı gösterime dönüştürme işlemi yapacağı zaman en yakına yuvarlama metodunu kullanmaktadır [100].

$$e = \sum_{i=0}^{w_e-1} e_i 2^i \tag{4.2}$$

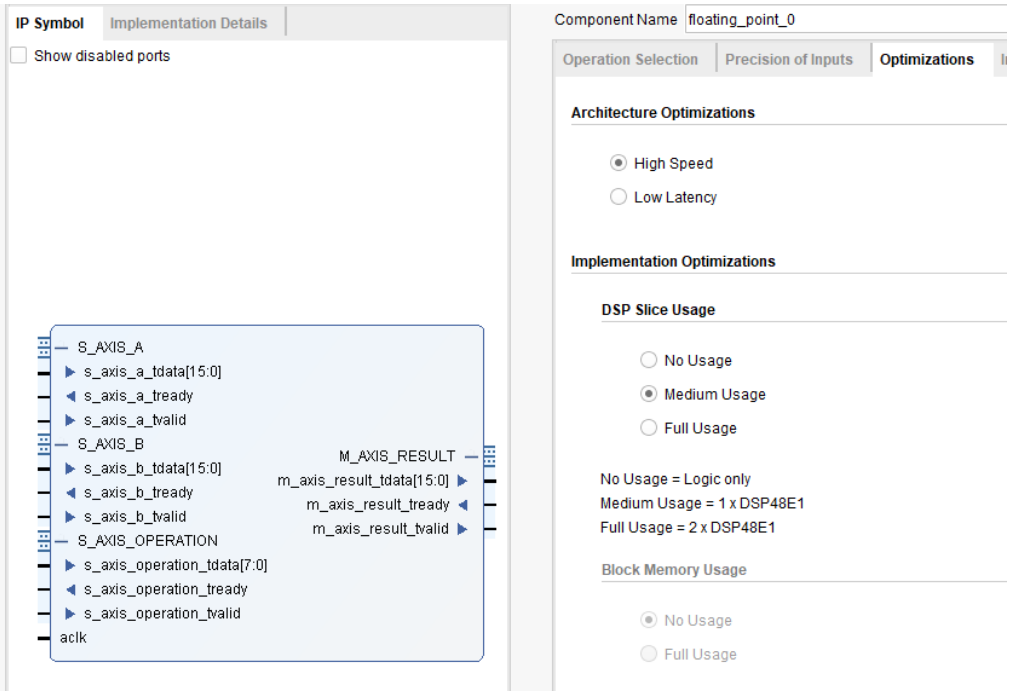
ve

$$w_e = w - w_f$$

$$E = e - (2^{W_e - 1} - 1) \tag{4.3}$$

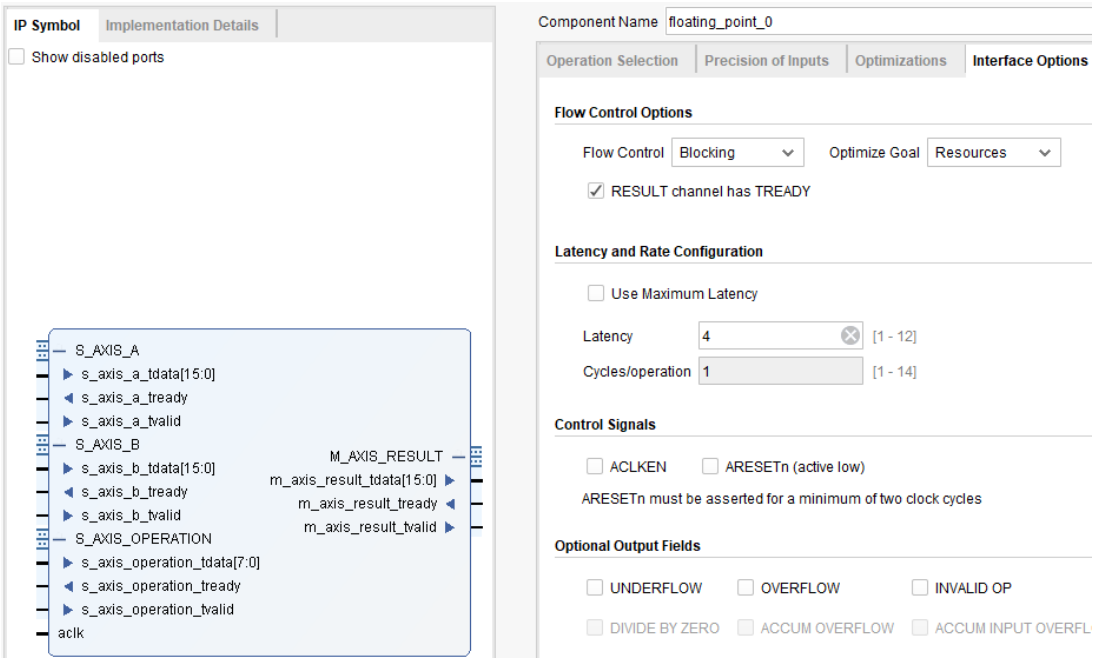
IP Çekirdek, kullanıcı ile AXI4-Stream [101] protokolü kanalları kullanılarak haberleşmektedir. Bir kanal her zaman TVALID ve TDATA'dan, ayrıca birkaç isteğe bağlı port ve alandan oluşmaktadır. Kayan Nokta Operatöründe desteklenen isteğe bağlı portlar TREADY, TLAST ve TUSER'dir. TVALID ve TREADY birlikte, yükün TDATA, TUSER ve TLAST olduğu bir mesajı aktarmak için bir el sıkışma gerçekleştirmektedir. Kayan Nokta Operatörü, TDATA alanlarında bulunan işlenenler üzerinde çalışır ve sonucu çıkış kanalının TDATA alanına çıkarmaktadır. Kayan Nokta Operatörü, TUSER ve TLAST girişlerini olduğu gibi kullanmamaktadır, ancak çekirdek, bu alanları TDATA için olduğu gibi aynı gecikmeyle iletme olanağı sağlamaktadır [100].

Kayan Noktalı Gösterim IP Çekirdeği, kaynak tüketimi açısından 3 farklı Dijital Sinyal İşlemcisi (Digital Signal Processor – DSP) kullanımı seçeneği ile çağırılabilir; sadece mantıksal birimler (Kullanım Yok), 1 tane DSP (Orta Kullanım) ya da 2 tane DSP (Tam Kullanım) şeklindedir. Blok hafıza kullanımı açısından ise hiç kullanmama ve tam kullanım seçenekleri bulunmaktadır. IP çekirdeğin bu şekilde sınıflandırılması FPGA içerisindeki kaynak tüketimi optimizasyonu açısından önemli bir rol oynamaktadır. Bu tez çalışmasında IP çekirdeklerin bir kısmı 1 adet DSP kullanacak şekilde, kalanı ise sadece mantık blokları kullanılacak şekilde üretilmiştir. Mimari optimizasyonu ise yüksek hız seçeneği ile oluşturulmuştur. Şekil 4.10.'da Kayan Noktalı Gösterim IP Çekirdeğinin IP Katalog üzerinde kaynak tüketiminin ayarlandığı sekme verilmiştir.




**Şekil 4.10.** Kayan Noktalı Gösterim IP Çekirdeği IP Katalog - Kaynak Tüketimi Sekmesi

Çekirdeğin aldığı girişe göre verdiği çıktı süresi de ayarlanabilen bir başka parametredir. Giriş-çıkış arasındaki süre FPGA'nın kullandığı saat çevrimi cinsindan belirtilmektedir. Bu sürenin optimizasyonu AMD Xilinx tarafından IP Çekirdeğinin farklı FPGA'ler ile üretildiğinde alabileceği maksimum saat çevrimi frekansı hesabına göre gerçekleştirilebilmektedir. Bu tez çalışmasında bu süre 2 saat çevrimi olarak belirlenmiştir. Şekil 4.11.'de IP Çekirdek IP Kataloğu üzerinden çekirdeğin vereceği çıktı süresinin ayarlandığı sekme gösterilmiştir. Tablo 4.1.'de ise IP Çekirdeğin Kintex-7 (XC7K70T) serisi bir FPGA üzerinde çağırılıp farklı operasyonlar ile kullandığı kaynak tüketimi ve alınabilecek en yüksek saat çevrimi frekansı bilgilerinin de içinde bulunduğu tablo verilmiştir.




**Şekil 4.11.** Kayan Noktalı Gösterim IP Çekirdeği IP Katalog - Gecikme Ayarları Sekmesi

**Tablo 4.1.** XC7K70T Serisi FPGA'da Kayan Noktalı Gösterim IP Çekirdeği Ayarlarına Göre Kaynak Tüketimi

| Operasyon | Hassasiyet Tipi | Kaynak Kullanımı | Blok RAM Kullanımı | Gecikme | $F_{max}$ (MHz) | LUT (Lookup Tablosu) | FF (Flip Flop) | DSP | 36k BRAM |
|---|---|---|---|---|---|---|---|---|---|
| Ekle-Çıkart | Yarı | Tam | Yok | - | 457 | 105 | 227 | 2 | 0 |
| Ekle-Çıkart | Yarı | Orta | Yok | - | 450 | 139 | 289 | 1 | 0 |
| Ekle-Çıkart | Yarı | Yok | Yok | - | 516 | 173 | 235 | 0 | 0 |
| Bölme | Yarı | - | Yok | 1 | 613 | 235 | 373 | 0 | 0 |
| Bölme | Yarı | - | Yok | 13 | 391 | 93 | 120 | 0 | 0 |
| Üstel | Yarı | - | Yok | - | 502 | 172 | 177 | 2 | 0 |
| Çarpma | Yarı | Tam | Yok | - | 457 | 52 | 90 | 1 | 0 |
| Çarpma | Yarı | Orta | Yok | - | 457 | 60 | 93 | 2 | 0 |
| Çarpma | Yarı | Yok | Yok | - | 487 | 161 | 206 | 0 | 0 |
| Karekök | Yarı | - | Tam | 1 | 457 | 26 | 38 | 0 | 1 |
| Karekök | Yarı | - | Yok | 1 | 583 | 141 | 233 | 0 | 0 |

# 5. UYGULANAN YÖNTEMLER

Bu çalışmada yürütülmüş olan veri eğitimi ve eğitime yönelik test ve deneyler, PyCharm 2024.1 Community Edition arayüzünde Python 3.10.13 ortamında Torch v2.3.1 ile gerçekleştirilmiştir. Python, günümüzde yapay zeka eğitimi, geliştirilmesi ve gerçeklenmesi adına çok yaygın bir şekilde kullanılan bir dildir. Ayrıca yüksek miktarda kütüphane envanteri ve kullanıcı kitlesi ile yapılan çalışmalardaki bilinmezliğin çözümlenmesinde kolaylık sağlamaktadır. Python aracılığı ile hazır birçok yapay zeka modeli kullanılabilmesine ek olarak kullanıcı tarafından da karmaşık sinir ağları tasarlanabilmekte ve bu model mimarileri üzerinde detaylı analizler yapılabilmektedir.

Pycharm, Python programlama için kullanılan bir tümleşik geliştirme ortamıdır (Integrated Development Environment - IDE). Python dilini terminalden kullanmak yerine bir arayüzden daha kullanışlı bir şekilde kullanıcıya sunan bir arayüzdür. Aynı zamanda içerisinde hata ayıklama, veri analizi, proje takibi ve kütüphane ortamları oluşturma gibi birçok kullanıcı dostu özelliğe sahiptir.

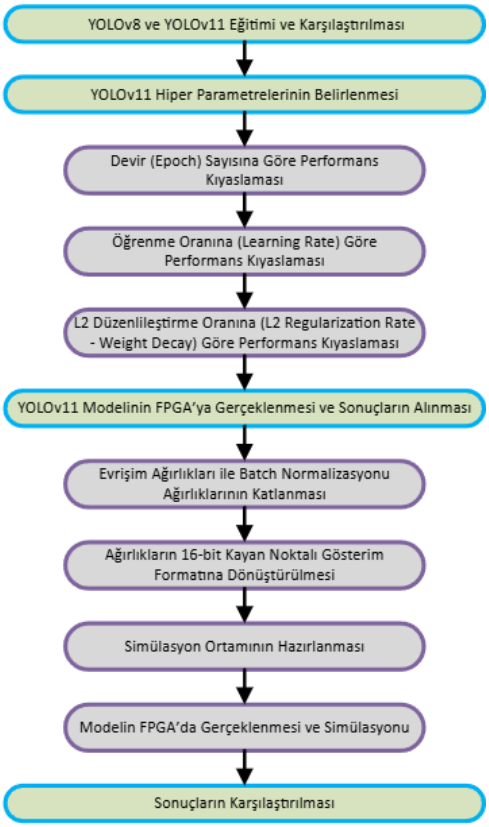
Bilgisayar ortamı, ortalamanın üstünde bir performansa sahiptir. Bileşenler olarak, AMD Ryzen9 5900x 12-Çekirdek @3.7-4.7 GHz işlemci, 16 GB RAM ve NVIDIA GeForce RTX 3070 Ti 8GB GPU (CUDA 11.8 ve cudNN v9.1 yazılımları ile destekli) ile büyük veri setleri ve karmaşık sinir ağı tabanlı modelleri üzerinde iyi bir performans sağlamaktadır. Eğitilen YOLOv11n modelinin gerçeklendiği ve simülasyonların yapıldığı ortam ise AMD Xilinx Vivado 2020.1 ortamıdır ve kullanılan dil ise VHDL (2001 versiyonu) dilidir. Gerçeklenen modelin simülasyonu ve entegrasyonu Kintex-7 serisi bir FPGA olan XC7K410T-FFG900 parça numaralı FPGA'sıdır.

Çalışmada değerlendirilen YOLO modelleri arasında YOLOv8n, YOLOv8s ve YOLOv11n bulunmaktadır. Bu mimariler, kendi içlerinde hiper parametreler sabit ve birbirleri ile aynı olacak şekilde performans karşılaştırması yapılmıştır. Performans karşılaştırmasından öncesinde ise modellerin ağırlık sayılarına bakılmıştır. Bu doğrultuda YOLOv8n 3,012,408 adet, YOLOv8s 11,138,696 adet ve YOLOv11n ise 2,591,400 adet ağırlığa sahiptir. Ağırlık sayısının az olması demek modelin entegrasyonu esnasında kullanılacak hafıza elementinin az olması demektir. Bu bağlamda 3 farklı yapay zeka model mimarisi için detection-pcb-defects-yolov8 veri seti kullanılarak en iyi ağırlık sayısı-

performans ilişkisi gösteren model seçilip, bu model için optimum hiper parametrelerin belirlenmesi hedeflenmiştir. Yapılan testlerde veri setinin %80'i eğitim, %10'u doğrulama ve %10'u test için ayrılmıştır.

Çalışmada seçilmiş olan YOLOv11n modelinin FPGA için RTL modellemesi gerçekleştirilmiştir. Modellemiş olan mimari gerçek zamanlamalara uygun bir video çıktı simülasyonu üzerinde test edilerek benzetim sonuçları Python ortamında çizdirilmiş ve doğruluğu hesaplanmıştır. Oluşturulan mimarinin doğruluğu %97,86 olarak hesaplanmıştır. Aynı zamanda bir video çerçevesinin ilk pikseli sisteme ulaştığı andan itibaren çıkarım (inference) süresi 34,9386306ms olarak benzetim sonuçlarında görülmüştür.

Çalışma toplamda 4 ana aşamadan oluşmaktadır. İlk aşamada kullanılacak modelin belirlenmesi adına modellerin eğitilip karşılaştırılması, ikinci aşamada seçilen modelin en başarılı sonucu vermesi için hiper parametrelerinin belirlenmesi, üçüncü aşamada seçilen modelin FPGA için uygun bir mimarisinin oluşturulup gerçeklenmesi ve sonuçların alınması ve son olarak alınan sonuçların Python ortamındaki sonuçlarla ve literatür ile hem hız hem de doğruluk olarak kıyaslanmasıdır. Şekil 5.1.'de çalışmada gerçekleştirilen süreç adımları ve alt adımları gösterilmiştir.




**Şekil 5.1.** Çalışmanın Adımları

## 5.1. YOLOv8 ve YOLOv11 Eğitimi ve Karşılaştırılması

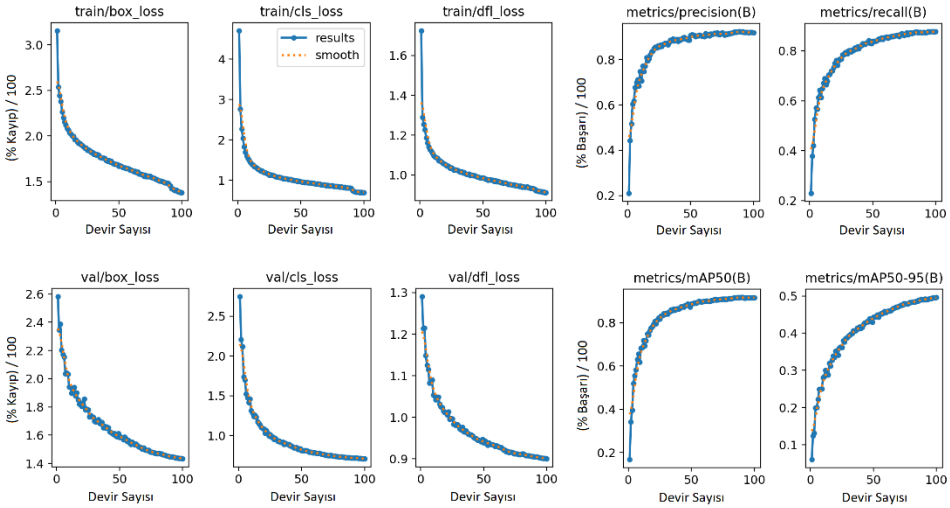
Bu çalışmada eğitim hiper parametreleri varsayılan değerler olarak tutulmuştur. YOLOv8 ve YOLOv11 varsayılan değerleri karşılaştırılmış ve farklılık olanlar eşitlenmiştir. Devir sayısı ise 100 olarak belirlenmiştir. Bu kapsam dahilinde mini yığın oranı 16, öğrenme oranı 0,01, momentum sayısı 0,937 ve ağırlık L2 düzenlileştirme (L2 Regularization – weight decay) oranı 0,0005 olarak ayarlanmıştır. Her mimari aynı veri seti ile eğitilmiş, sadece optimizasyon yönteminde değişiklikler yapılmıştır. Kullanılan optimizasyon yöntemleri; Adam, SGD, AdamW, NAdam, RAdam, RMSPRop yöntemleridir. Eğitim sonuçları, mAP50 cinsinden ve en iyi eğitim sonucunun alındığı devir Tablo 5.1.'de paylaşılmıştır. Şekil 5.2.'de YOLOv8n mimarisinin, Şekil 5.3.'te YOLOv8s mimarisinin, Şekil 5.4.'te ise YOLOv11n mimarisinin en iyi sonuçlarının alındığı eğitimin grafikleri paylaşılmıştır.

Karşılaştırılan bütün YOLO mimarilerinde alınan sonuçlarda en iyi sonucu veren "SGD" optimizasyon yöntemidir. En iyi sonuç alınan SGD optimizasyon yönteminde ise YOLOv8n mimarisi %91,5, YOLOv8s %94 ve YOLOv11n %93,6 mAP başarısına sahiptirler. Diğer optimizasyon yöntemlerinde genel olarak YOLOv8s mimarisi diğer mimarilere göre yaklaşık %4'lük daha fazla eğitim başarısına sahiptir. En kötü sonuçlar alınan RMSProp optimizasyon yönteminde ise üç model de diğer yöntemlere göre çok daha düşük performans göstermiştir. RMSProp haricinde bütün optimizasyon yöntemlerinde en iyi sonuçlar toplam devir sayısının son %10'luk diliminde yer almaktadır. Bu yüzden RMSProp dışındaki diğer optimizasyon yöntemleri daha fazla devir sayısı ile eğitilerek daha iyi sonuçlar alınabileceği ön görülmüştür.

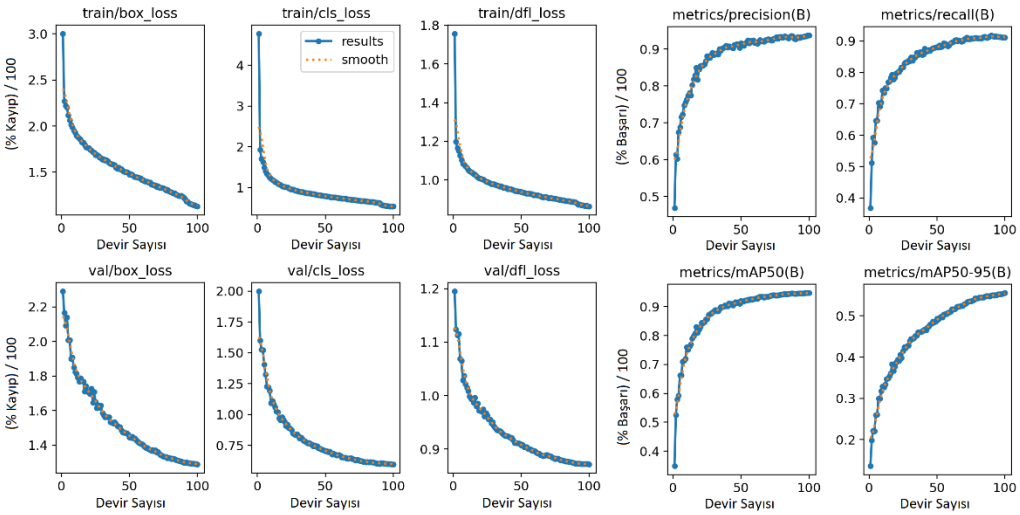
Bu deneyde YOLO mimarileri arasından en iyi performansı YOLOv8s modeli göstermiştir ancak en iyi gösterilen performans veya diğer optimizasyon yöntemlerindeki performans farkı ile sahip olunan eğitim başarısı kıyaslandığı zaman YOLOv11n modelinin YOLOv8 varyantlarına karşı daha yüksek ağırlık sayısı-performans ilişkisine sahip olduğunu söylemek mümkündür. Bu sebeple bu tez çalışmasında YOLOv11n modelinin kullanılmasına karar verilmiştir.

**Tablo 5.1.** Farklı YOLO Modellerinin Eğitim Kıyaslaması

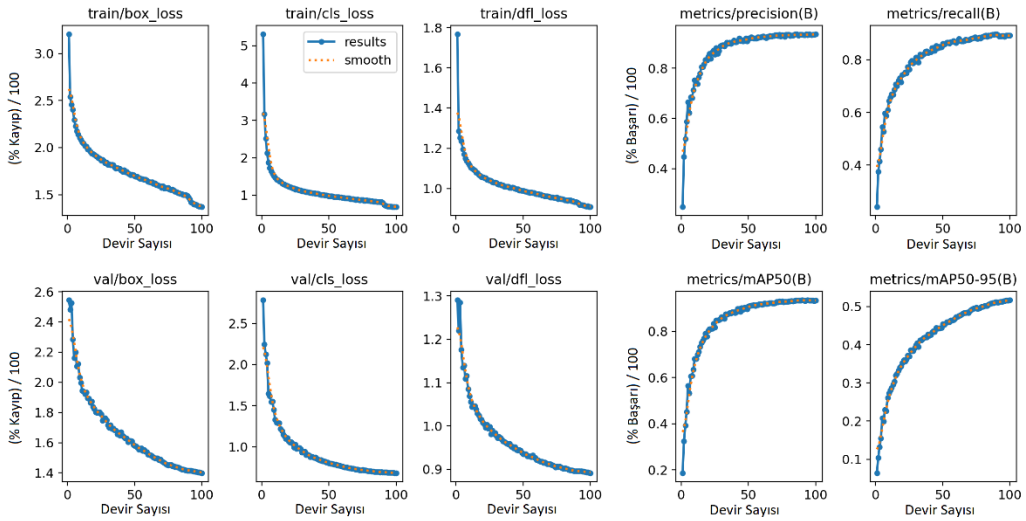
| Model | Optimizasyon Yöntemi (Mini Yığın = 16, Epoch = 100, Momentum = 0,936, Lr = 0,01, W_Decay = 0,0005) | | | | | |
|---|---|---|---|---|---|---|
| | **Adam** | **AdamW** | **NAdam** | **RAdam** | **SGD** | **RMSProp** |
| **YOLOv8n** | 0,788 (Devir 92) | 0,851 (Devir 100) | 0,826 (Devir 100) | 0,824 (Devir 97) | **0,915 (Devir 100)** | 0,315 (Devir 47) |
| **YOLOv8s** | 0,823 (Devir 90) | 0,881 (Devir 100) | 0,849 (Devir 100) | 0,842 (Devir 98) | **0,940 (Devir 100)** | 0,501 (Devir 81) |
| **YOLOv11n** | 0,782 (Devir 90) | 0,856 (Devir 100) | 0,823 (Devir 99) | 0,816 (Devir 99) | **0,936 (Devir 98)** | 0,455 (Devir 73) |




**Şekil 5.2.** YOLOv8n En İyi Eğitim Sonuç Grafikleri




**Şekil 5.3.** YOLOv8s En İyi Eğitim Sonuç Grafikleri




**Şekil 5.4.** YOLOv11n En İyi Eğitim Sonuç Grafikleri

## 5.2. YOLOv11 Hiper Parametrelerinin Belirlenmesi

Bu çalışmada yapılan deneylerde YOLOv11n modeli ile devam edilmiştir. YOLOv11n modelinin en uygun biçimde eğitilebilmesi için gereken hiper parametre değerlerinin bulunması amaçlanmıştır. Uygun değerlerinin veya tiplerinin bulunması amaçlanan hiper parametreler optimizasyon yöntemi, devir sayısı, öğrenme oranı ve L2 düzenlileştirme oranıdır (ağırlık bozulması – weight decay). Mini yığın oranı varsayılan değer olan 16'da tutulmuştur. Bunun sebebi ise eğitim esnasında veri kümesinin 64'e ve 32'ye bölünmesi gradyan hesaplaması ve ağırlık değerlerinin güncellenme işlemlerini 16'ya bölünmesine göre daha az veriyle yapması sebebiyet vermektedir. Bu durum işlem oranını azaltarak eğitim süresinin azalmasını fakat kayıp değerinde artışa ve doğruluk oranında ise düşüşe sebep olmaktadır. Bu kapsam dahilinde optimizasyon yöntemleri olarak Adam, SGD, AdamW, NAdam, RAdam ve RMSPRop optimizasyon yöntemlerinin hepsi en az 100 ve 200 devir, bazıları ise ek olarak 400 ve 600 devir eğitilerek performans kıyaslaması yapılmıştır. Öğrenme oranının tespitinde ise 0,05, 0,01 ve 0,001 öğrenme oranları ile veri seti eğitilerek performans kıyaslaması yapılmıştır. Son olarak L2 düzenlileştirme oranının tespiti için çalışma yapılmış, YOLOv11n modeli 0,0001, 0,0005 ve 0,0009 oranları ile eğitilmiştir. Yapılan bütün eğitimlerde sadece bulunması hedeflenen parametre değiştirilmiş, diğer bütün parametreler sabit tutulmuştur.

Yapılan bütün testler ve karşılaştırmalar dahilinde optimizasyon yöntemleri arasından kendisinden sonraki en başarılı eğitime göre %10 daha başarılı mAP50 değeri yakalayarak %98,4 mAP50 başarısı ile 600 devirle eğitilen SGD optimizasyon yöntemi en iyi sonuçları

vermiştir. Öğrenme oranı için yapılan testlerde ise 0.01 öğrenme oranı ile eğitilen model %98,4 mAP50 başarısı ile en iyi performansı göstermiştir. L2 düzenlileştirme oranına yönelik yapılan testlerde 0.0005 L2 düzenlileştirme oranı verilerek eğitilen YOLOv11n modeli %98,4 mAP50 başarısı ile en yüksek performansı göstermiştir.

Bu deneyler sonucunda farklı parametreler arasından en iyi performansı gösteren konfigürasyon YOLOv11n modelinin 600 devirde SGD optimizasyon yöntemi ile 0,01 öğrenme oranında ve 0,0005 L2 düzenlileştirme oranı ile eğitilmesi ile gözlemlenmiştir ve YOLOv11n modelinin bu parametreler ile eğitilmesine karar verilmiştir.

### 5.2.1. Devir (Epoch) Sayısına Göre Performans Kıyaslaması

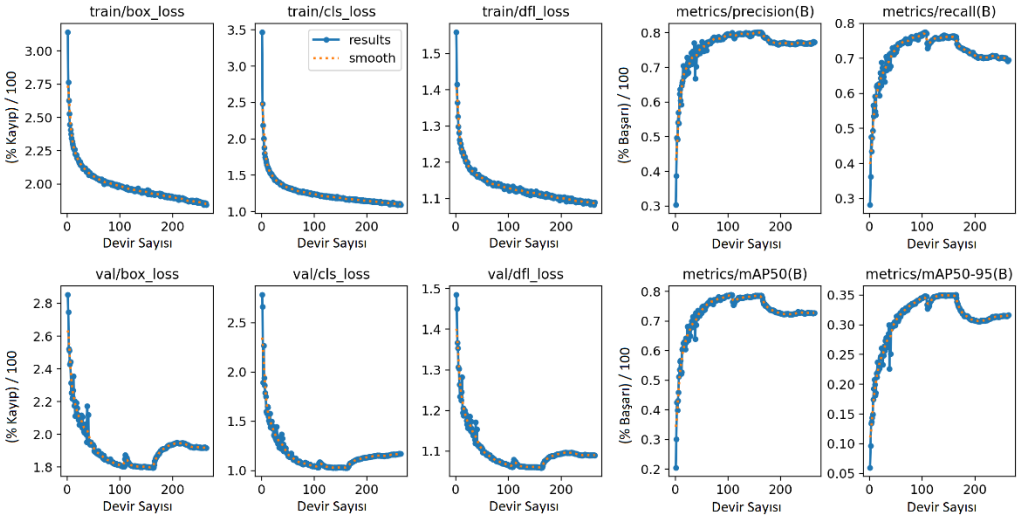
Bu çalışmada yapılan deneylerde YOLOv11n'in farklı optimizasyon yöntemlerinin farklı devir sayıları ile eğitilerek performans karşılaştırması amaçlanmıştır. Mini yığın boyutu varsayılan değer olan 16 olarak ayarlanmıştır. Momentum, öğrenme oranı ve ağırlık bozulma oranı da varsayılan değerleri olan 0,937, 0,01 ve 0,0005 olarak ayarlanmıştır. Her bir optimizasyon yöntemi ise devir sayısı sırasıyla 100, 200, 400 ve 600 olacak şekilde üç noktada incelenmiştir. Yapılan bu deney sonucunda en iyi performans alınan optimizasyon yöntemi ve eğitilecek devir sayısına karar verilmiştir. Eğitim sonuçları, mAP50 cinsinden ve en iyi eğitim sonucunun alındığı devir Tablo 5.2.'de paylaşılmıştır. Şekil 5.5.'te Adam optimizasyon yönteminin, Şekil 5.6.'da AdamW optimizasyon yönteminin, Şekilde 5.7.'de NAdam optimizasyon yönteminin, Şekil 5.8.'de RAdam optimizasyon yönteminin, Şekil 5.9.'da SGD optimizasyon yönteminin, Şekil 5.10.'da ise RMSProp optimizasyon yönteminin en iyi sonuçlarının alındığı eğitimin grafikleri paylaşılmıştır.

Testlerin yapıldığı optimizasyon yöntemlerinden Adam en fazla %78,7, AdamW %88,4, NAdam %82,9, RAdam %82,8, SGD %98,4 ve RMSProp ise %46,1 mAP50 başarısı göstermişlerdir. RMSProp optimizasyon yönteminde 100 ve 200 devirlerde alınan düşük performanstan dolayı 400 ve 600 devir eğitimi zamandan tasarruf etmek adına bu optimizasyon yöntemi için uygulanmamıştır. SGD optimizasyon yöntemi dışındaki diğer optimizasyon yöntemleri için ise 400 devire gelmeden çok öncesinde başarıları ve kayıp değerleri sabit noktaya ulaştığı ve SGD'ye göre düşük başarı gösterdikleri için 600 devir eğitimleri gerçekleştirilmiştir. SGD optimizasyon yöntemi için kayıp değerleri 400 devirde sabit bir noktaya ulaşmadığından dolayı 600 devir eğitimi de gerçekleştirilmiş, mAP50 değeri ve kayıp değerlerinin neredeyse sabit bir noktada seyrettiği tespit edilmiştir.

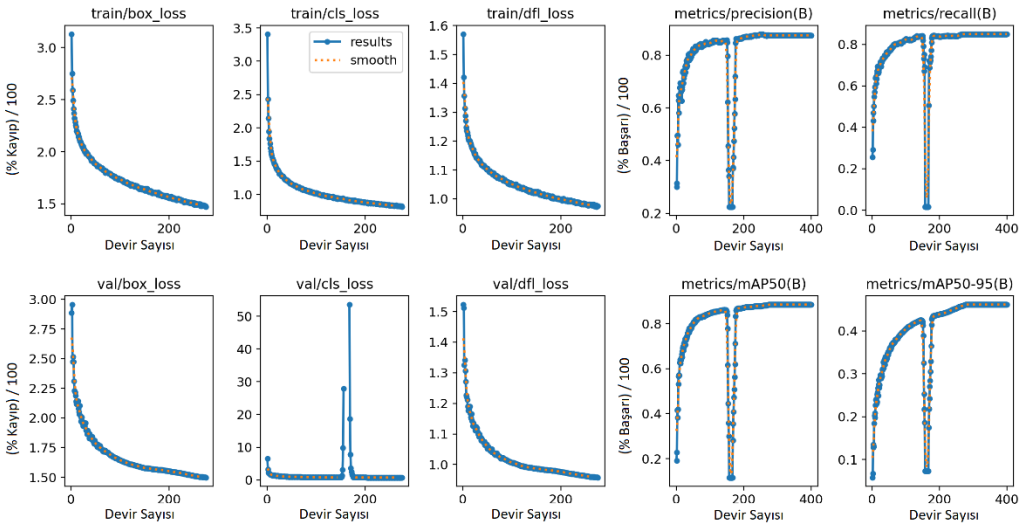
Yapılan bu deney sonucunda optimizasyon yöntemleri arasında en yüksek başarıyı 600 devir ile eğitilen SGD optimizasyon yöntemi göstermiştir. Bu başarı sebebiyle optimizasyon yöntemi olarak SGD optimizasyon yöntemi, eğitim esnasında kullanılacak devir sayısı olarak ise 600 devir kullanılmasına karar verilmiştir.

**Tablo 5.2.** Farklı Optimizasyon Yöntemlerinin Farklı Devirlerdeki Performansı

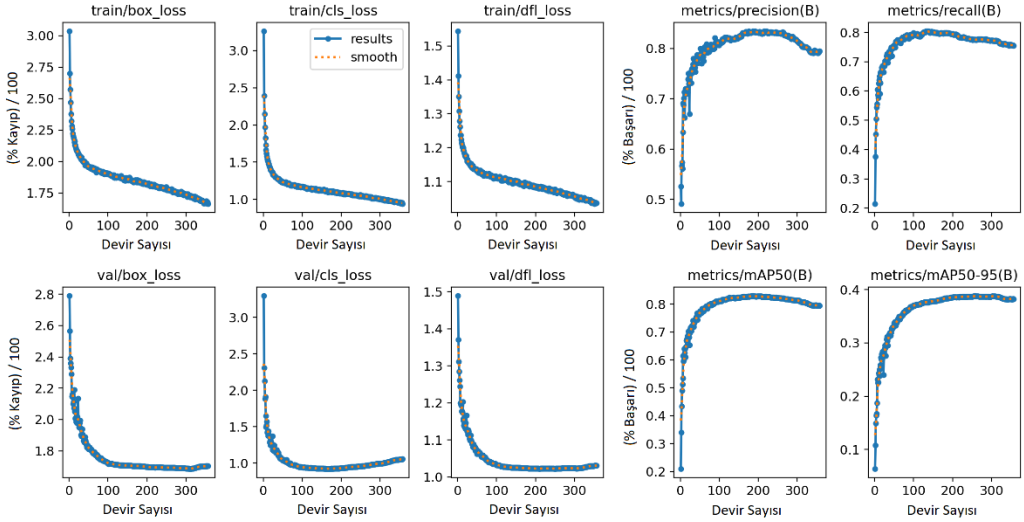
| **Optimizasyon Yöntemi** | **Devir Sayısı (Mini Yığın = 16, Momentum = 0,936, Lr = 0,01, W_Decay = 0,0005)** | | | |
|---|---|---|---|---|
| | **100 Devir** | **200 Devir** | **400 Devir** | **600 Devir** |
| **Adam** | 0,782 (Devir 90) | 0,786 (Devir 169) | 0,787 (Devir 173) | Eğitilmemiştir. |
| **AdamW** | 0,856 (Devir 100) | 0,868 (Devir 199) | 0,884 (Devir 276) | Eğitilmemiştir. |
| **NAdam** | 0,823 (Devir 99) | 0,825 (Devir 196) | 0,829 (Devir 219) | Eğitilmemiştir. |
| **RAdam** | 0,816 (Devir 99) | 0,827 (Devir 192) | 0,828 (Devir 320) | Eğitilmemiştir. |
| **SGD** | **0,936 (Devir 98)** | **0,956 (Devir 200)** | **0,974 (Devir 397)** | **0,984 (Devir 580)** |
| **RMSProp** | 0,455 (Devir 73) | 0,461 (Devir 166) | Eğitilmemiştir. | Eğitilmemiştir. |



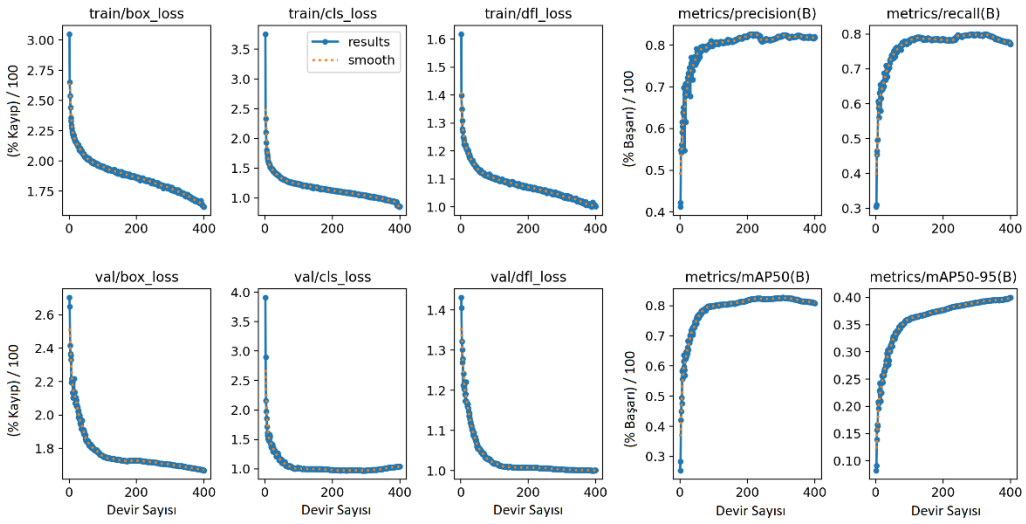

**Şekil 5.5.** Adam Optimizasyon Yöntemi Eğitim Sonuçları



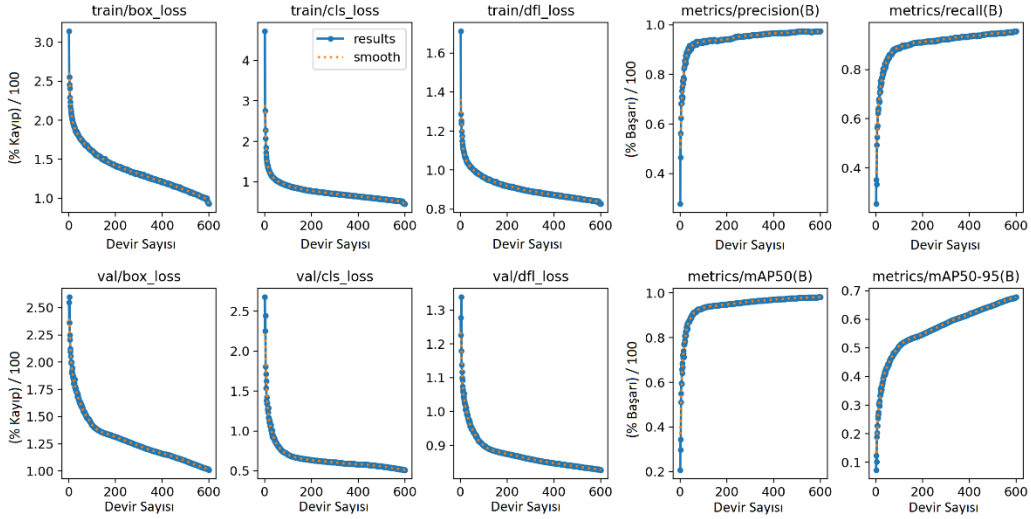

**Şekil 5.6.** AdamW Optimizasyon Yöntemi Eğitim Sonuçları



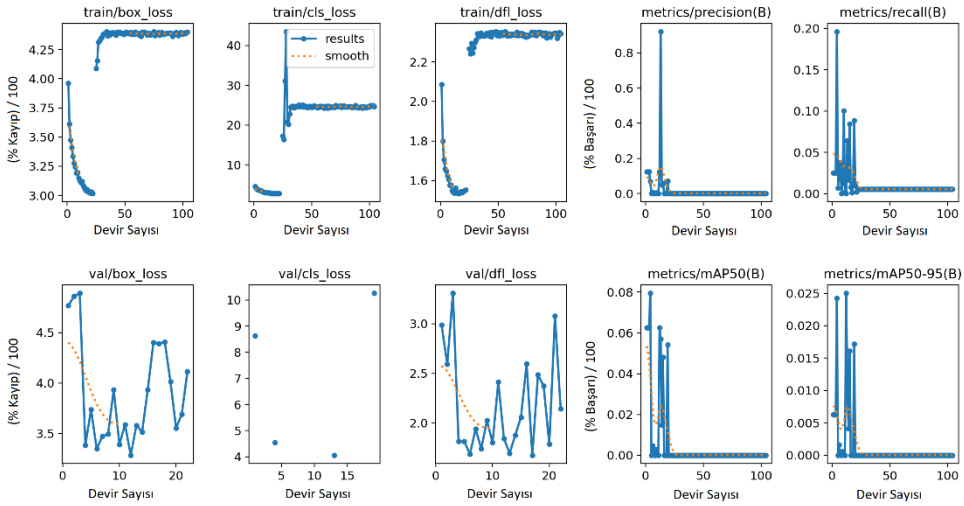

**Şekil 5.7.** NAdam Optimizasyon Yöntemi Eğitim Sonuçları




**Şekil 5.8.** RAdam Optimizasyon Yöntemi Eğitim Sonuçları




**Şekil 5.9.** SGD Optimizasyon Yöntemi Eğitim Sonuçları




**Şekil 5.10.** RMSProp Optimizasyon Yöntemi Eğitim Sonuçları

### 5.2.2. Öğrenme Oranına (Learning Rate) Göre Performans Kıyaslaması

Bu aşamada yapılan deneylerde SGD optimizasyonu ve 600 devir sayısı ile devam edilmiştir. Mini yığın boyutu, momentum ve ağırlık bozulma oranı da varsayılan değerleri olan 16, 0,937 ve 0,0005 olarak ayarlanmıştır. Üç farklı öğrenme oranı 0,05, 0,01 ve 0,001 ile veri seti eğitilmiştir. Yapılan deney sonucunda aralarında en iyi performansı gösteren öğrenme oranına karar verilmiştir. Eğitim sonuçları, mAP50 cinsinden ve en iyi eğitim sonucunun alındığı öğrenme oranı Tablo 5.3.'te paylaşılmıştır. En iyi performansa sahip eğitim sonuçlarını içeren grafikler Şekil 5.9.'da paylaşılmış olan grafiklerdir.

Veri seti 0,05 öğrenme oranı ile eğitildiğinde %93,8'lik bir başarıya ulaşmıştır. Eğitim esnasında elde edilen kayıp değerlerindeki oynamalar artı ve eksi yöne doğru çok fazladır miktarda gözlemlenmiştir, 0.01 öğrenme oranı ile eğitildiğinde %98,4 başarı ile yüksek bir performans artışı gözlemlenmiş, 0.001 öğrenme oranı ile eğitildiğinde ise %1,6'lık bir performans düşüşü göstererek %96,8 mAP50 başarı değerine ulaştığı gözlemlenmiştir. 0,001 öğrenme oranı ile eğitim ancak daha fazla devir sayısı ile eğitim yapıldığı zaman artış göstereceği ön görülmüştür.

Öğrenme oranını belirlemeye yönelik yapılan deneyler sonucunda en iyi sonuçlar 0,01 öğrenme oranında alınmış ve bu öğrenme oranının eğitim hiper parametresi olarak seçilmesine karar verilmiştir.

**Tablo 5.3.** Farklı Öğrenme Oranlarındaki Eğitim Performansı

| **Öğrenme Oranı (Mini Yığın = 16, Momentum = 0,936, Devir = 600, W_Decay = 0,0005)** | | |
|---|---|---|
| **Lr = 0,05** | **Lr = 0,01** | **Lr = 0,001** |
| 0,938 (Devir 368) | **0,984 (Devir 580)** | 0,968 (Devir 596) |

### 5.2.3. L2 Düzenlileştirme Oranına (L2 Regularization Rate - Weight Decay) Göre Performans Kıyaslaması

Son deney olarak ise L2 düzenlileştirme oranının belirlenmesi adına çalışma yapılmıştır. Önceki deneylerde karar verilen devir ve öğrenme oranı olarak 600 devir ve 0,01 öğrenme oranı kullanılmıştır. Mini yığın boyutu ve momentum varsayılan değerlerinde bırakılarak 16 ve 0,937 olarak ayarlanmıştır. Üç farklı L2 düzenlileştirme oranı 0,0001, 0,0005 ve 0,0009 eğitim parametresi olarak verilerek veri seti eğitilmiştir. Yapılan deney sonucunda en iyi performansı gösteren düzenlileştirme oranının eğitim parametresi olarak seçilmesine karar verilmiştir. Eğitim sonuçları, mAP50 cinsinden ve en iyi eğitim sonucunun alındığı düzenlileştirme oranı Tablo 5.4.'te paylaşılmıştır. En iyi performansa sahip eğitim sonuçlarını içeren grafikler Şekil 5.9.'da paylaşılmış olan grafiklerdir.

Eğitim sonuçları olarak 0.0001 L2 düzenlileştirme oranına sahip eğitimin mAP50 başarısı %98,1, 0.0005 için %0,3'lük bir artış gözlemlenerek %98,4 ve 0,0009 için devir 586'dan sonrasına düşüşe geçmeye başlayarak ve %0,4 düşüş göstererek %98 olarak gözlemlenmiştir.

**Tablo 5.4.** Farklı L2 Düzenlileştirme Oranlarındaki Eğitim Performansı

| **L2 Düzenlileştirme Oranı (Weight Decay) (Mini Yığın = 16, Momentum = 0,936, Lr = 0,01 Devir = 600)** | | |
|---|---|---|
| **w_decay = 0,0001** | **w_decay = 0,0005** | **w_decay = 0,0009** |
| 0,981 (Devir 573) | **0,984 (Devir 580)** | 0,98 (Devir 586) |

## 5.3. YOLOv11 Modelinin FPGA'ya Gerçeklenmesi ve Sonuçların Alınması

Bu çalışmada YOLOv11n derin öğrenme modelinin FPGA entegrasyonuna uygun bir hale getirilebilmesi için izlenen adımlar paylaşılmıştır. YOLOv11 modeli kendi mimarisi

gereğince Batch Normalizasyonu katmanı içermektedir. Bu katman işlem karmaşıklığı ve hafıza tüketimi yüksek bir katmandır. Derin öğrenme uygulamalarında evrişim katmanının ve normalizasyon katmanının öğrenmiş olduğu bir ağırlık seti olduğu için ve evrişim katmanında ağırlıklar ile girdi pikseller çarpılacağından dolayı normalizasyon katmanı ağırlık ve önyargı değerleri evrişim katmanı ağırlıkları içerisine dağıtılabilmektedir. Bu operasyona katlama (folding) operasyonu adı verilmektedir. Bu sayede işlem karmaşıklığı azalarak sadece çarpma ve toplama işlemi barındıran bir hale bürünmektedir. Bu yüzden öncelikle Batch Normalizasyonu katmanı ile evrişim katmanı birleştirilerek ağırlıklar yeniden hesaplanmıştır. Ağırlıklar yeniden hesaplandıktan sonrasında FPGA mimarisinin de çalışacağı format olan 16-bit kayan noktalı gösterim formatına dönüştürülerek bir ".coe" uzantılı hafıza ilklendirme dosyası formatında kayıt altına alınmıştır.

Ön operasyonlar tamamlandıktan sonrasında FPGA mimarisi entegrasyonuna geçilmeden öncesinde simülasyon ortamı oluşturulmuştur. Simülasyon ortamının gerçek değerleri taşıması adına görüntü video girdisi sağlayan modüller oluşturulmuştur. Bu modüllerin ürettiği görüntü çerçevesi, formatı ve zamanlamaları VESA DMT v1.13 [102] görüntü standardına uygun olarak gerçekleştirilmiş ve video özellikleri olarak ise 1024x768 boyutlarında 60Hz video zamanlamaları seçilmesine karar verilmiştir.

Simülasyon ortamı oluşturulduktan sonrasında FPGA mimarisi oluşturulmuştur. FPGA içerisinde gerçekleştirilen her bir operasyon modüler bir şekilde mimarisi çizilerek anlatılmıştır. FPGA alınan her bir kanalı 8 bit olan 24 bitlik piksel verisinin 16-bit kayan noktalı gösterime dönüştürülme adımları gösterilmiştir. FPGA mimarisinde YOLOv11n modeline ait operasyonlar ise 2 adımlı 1 dolgulu 3x3 evrişim operasyonu, 1 adımlı 1 dolgulu 3x3 evrişim operasyonu, 1 adımlı 1x1 evrişim operasyonu, 5x5 maksimum havuzlama operasyonu, üst örnekleme operasyonu, DFL operasyonu, sınırlayıcı kutuların koordinatlarını ölçeklendirme ve sınıf skorlarını düzenleme operasyonlarıdır.

Oluşturulan mimarinin simülasyon sonuçları paylaşılmış, verilen çıktılara göre test veri seti üzerinde sistemin FPGA doğruluğu ve zamanlamaları hesaplanmıştır. Alınan doğruluk ve zamanlama sonuçlarına göre FPGA mimarisinin tespit doğruluğu %97,86 tahmin (inference) süresi ise görüntü çerçevesinin ilk pikseli sisteme iletildiği andan itibaren 34,9386306ms'dir.

### 5.3.1. Evrişim Ağırlıkları ile Batch Normalizasyonu Ağırlıklarının Katlanması

Yapay sinir ağlarında Batch Normalizasyonu katlama operasyonu, sinir ağlarının tahmin işlemini gerçekleştirmesi için hazırlanırken gerçekleştirilen bir optimizasyondur. Özellikle FPGA gibi matematiksel operasyonların yapılmasının zor olduğu cihazlarda önemli bir yere sahiptir. Batch Normalizasyonu Katlamasındaki temel amaç batchnorm katmanı ile evrişim katmanının ağırlık ve önyargı değerlerini birleştirerek Batch Normalizasyon katmanını ortadan kaldırıp etkisini korumaktır. Bu sayede ortalama alma ve standart sapma hesaplama adımları ortadan kalkmış olur. Batch Normalizasyon formülü x değeri giriş, μ değeri ortalama, $\sigma^2$ değeri varyans, ε sayısal kararlılık için sabit, γ Batch Normalizasyonu ağırlığı ve β ise önyargı değerleri olmak üzere Eşitlik 5.1'de gösterilmiştir.

$$y = \gamma \frac{x - \mu}{\sqrt{\sigma^2 + \varepsilon}} + \beta \tag{5.1}$$

Batch normalizasyonu formülünde giriş yerine konvolüsyon ağırlığı koyularak katlanmış değerlere sahip yeni evrişim ağırlıkları elde edilebilmektedir. Ayrıca bu çalışmada eğitilen modelin evrişim katmanında önyargı bulunmadığından dolayı ortalama ve önyargı değerleri sıfır olmuştur. Bu sebeple Batch Normalizasyon önyargı değerleri herhangi bir değişime tabi tutulmadan yeni katlanmış değerlere sahip evrişim katmanının ön yargı değerleri haline gelmektedir. W değeri evrişim katmanı ağırlığı, $W_k$ katlanmış ağırlık değeri, b önyargı ve $b_k$ katlanmış önyargı değeri olmak üzere Eşitlik 5.2'de katlanmış ağırlıkların hesaplanması için, Eşitlik 5.3'te ise katlanmış önyargı değerlerinin hesaplanması için kullanılan formüller gösterilmiştir.

$$W_k = \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} \mathrm{W} \tag{5.2}$$

$$b_k = \frac{\gamma}{\sqrt{\sigma^2 + \varepsilon}} (b - \mu) + \beta \tag{5.3}$$

### 5.3.2. Ağırlıkların 16-bit Kayan Noktalı Gösterim Formatına Dönüştürülmesi

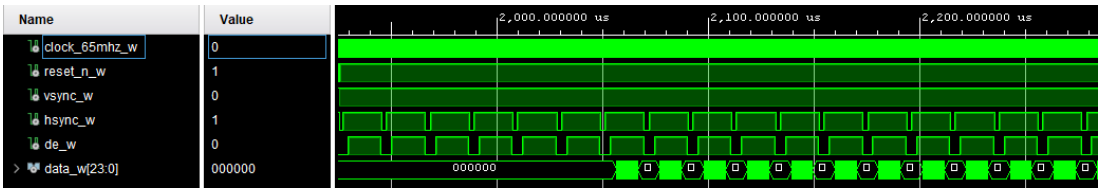
Ağırlıklar Python ortamında Batch Normalizasyonu operasyonuna tabi tutulduktan sonrasında 16-bit kayan noktalı gösterime dönüştürülmüştür. Dönüştürülen ağırlıkların işaret, üs ve mantissa parçalarının sırası korunacak şekilde on altılık tabana çevrilmiştir. On altılık tabana çevrilen bu ağırlık değerleri ise AMD Xilinx Vivado platformunda oluşturulan Blok RAM'ler için bir ilklendirme dosyası olan ".coe" uzantılı dosyalar halinde kaydedilmiştir.

Vivado platformunda oluşturulan Blok RAM'lerin 5.3.3. başlığında anlatıldığı üzere 3 parçaya bölünmesi sebebiyle ağırlıklar da 3 ana parçaya bölünmüş olup, her biri farklı ".coe" uzantılı ilklendirme dosyalarının içerisine kaydedilmiştir.

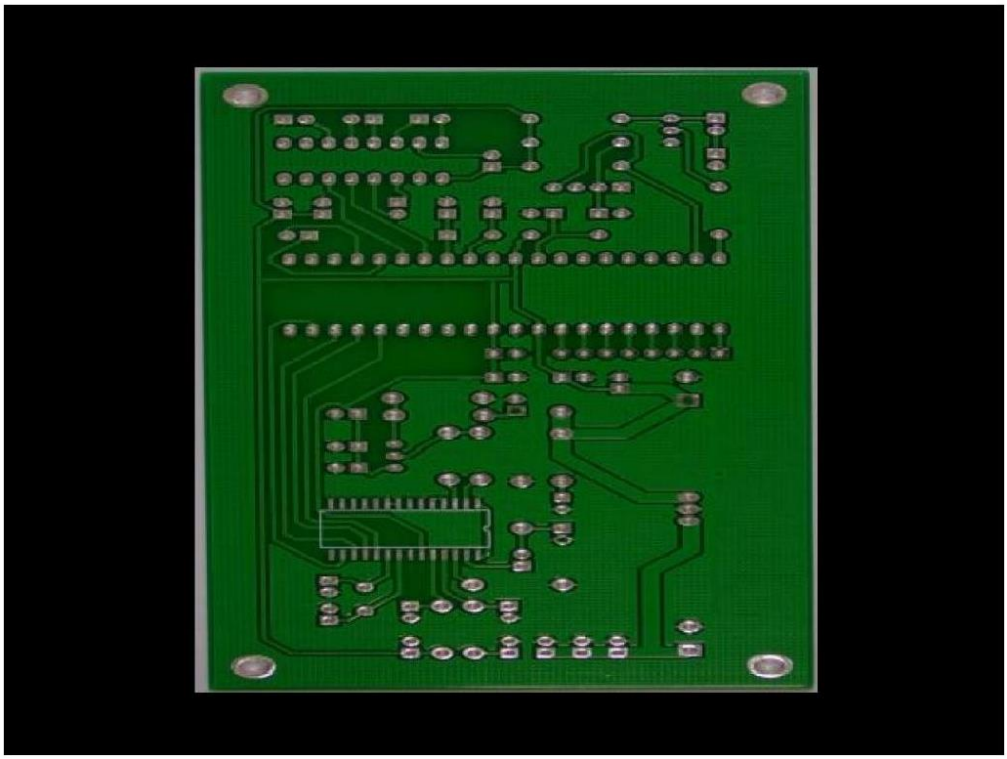
### 5.3.3. Simülasyon Ortamının Hazırlanması

Simülasyon ortamı, bir FPGA modelinin gerçek dünyaya entegrasyonu gerçekleştirilmeden önce sistemin davranışının ve zamanlamasının incelenebildiği, yazılan kod içerisindeki bütün sinyal akışlarının görülebildiği bir ortamdır. Bir diğer deyişle sıra tabanlı bir yazılım dilindeki hata ayıklama ortamıdır. Aynı zamanda simülasyondan alınan sonuçlar bir dosyaya yazılabilir ve başka bir program aracılığı ile de incelenebilir.

Çalışmanın test edilmesi için kurulan simülasyon ortamının dış dünya ile uyumlu olması adına VESA DMT v1.13 [102] standardı ile uyumlu 1024x768 60Hz video üretebilen bir video jeneratörü modülü oluşturulmuştur. Video jeneratörü tarafından oluşturulan video çerçevesi tamamen siyah piksellerden oluşmaktadır. Bu sebeple veri setinin test için ayrılmış olan 640x640 boyutlarındaki görüntülerin bu çerçevenin ortasına yerleştirilmesi için bir tane şablon (pattern) jeneratör modülü oluşturulmuştur. Şablon jeneratör modülünün çerçevenin ortasına ekleyeceği görseli elde edebilmesi için görsel bir ROM'a ".coe" uzantılı ilklendirme dosyası içerisinde gömülmüştür. ".coe" uzantılı dosyanın oluşturulabilmesi için MATLAB 2019a platformu kullanılmıştır. Video üretmek için kurulan yapı simülasyon ortamında çalıştırılarak test edilmiştir. Test sonucunda piksel verisinin iletildiği noktalar bir ".txt" uzantılı dosyaya yazılarak MATLAB ortamında yeniden görsele dönüştürülmüştür. Şekil 5.11.'de simülasyon görüntüsü, Şekil 5.12.'de ise yeniden oluşturulan görselin MATLAB sonucu verilmiştir.



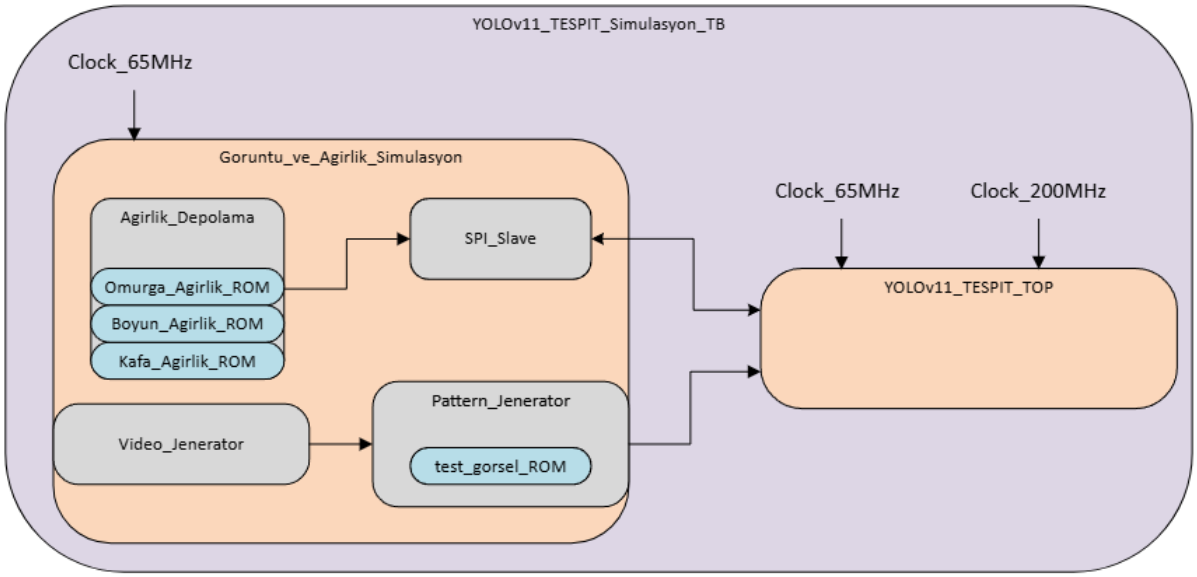

**Şekil 5.11.** Video Jeneratörü Simülasyonu



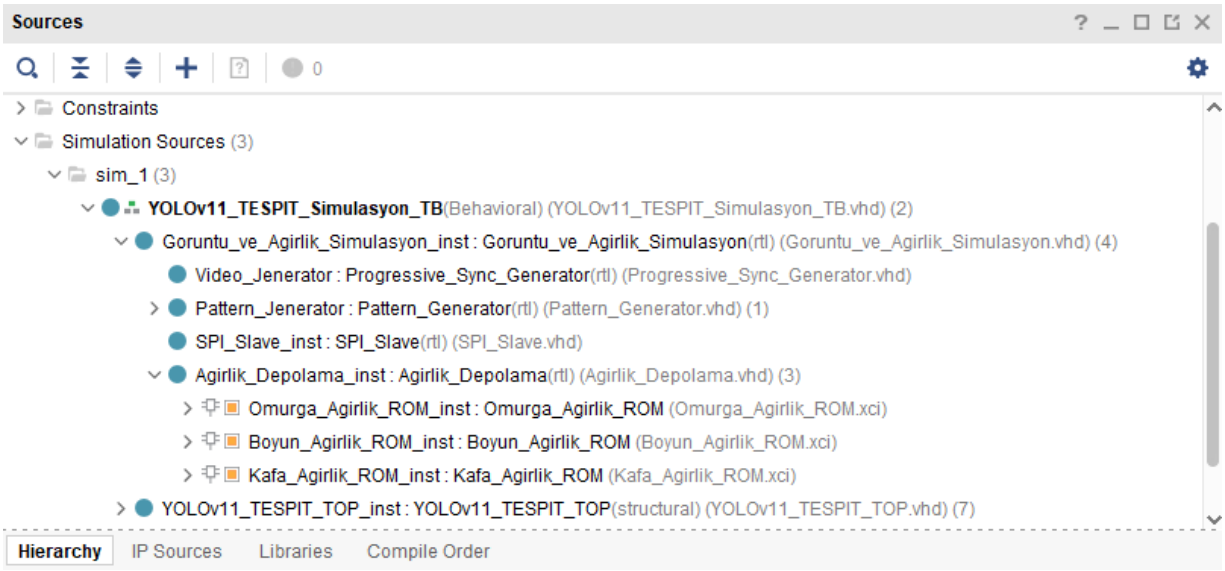
**Şekil 5.12.** Video Jeneratörü Sonuç Görsel

Simülasyon ortamına dahil edilen bir diğer modül ise eğitilmiş YOLOv11 modelinin ağırlıklarını içeren ve bir SPI (Serial Peripheral Interface – Seri Çevresel Arayüz) haberleşme protokolü yardımı ile test edilecek modüle istendiği zaman ağırlığı iletecek olan modüldür. Bu modül içerisinde ağırlıklar ROM'larda tutulmaktadır ve her bir ağırlık bir ROM adresinde yer almaktadır. AMD Xilinx Vivado platformunda ip çekirdek kataloğundan ROM oluşturulurken yazılabilecek maksimum adres sayısı 1.048.576 olduğundan dolayı evrişim ağırlıkları üç ana parçaya bölünmüştür; omurga ağırlıkları, boyun ağırlıkları ve kafa ağırlıkları. Her ana parçanın ağırlıklarının yüklendiği üç farklı ROM oluşturularak bu ağırlıklar ".coe" uzantılı ilklendirme dosyaları aracılığı ile ROM'ların içerisine aktarılmıştır.

Oluşturulan simülasyon ortamı YOLOv11n modelinin gerçeklendiği ana modüle bağlanacak şekilde sinyaller simülasyon modülünün dışarısına çıkılmıştır. Modüllerin kullanacakları 65MHz ve 200MHz frekansına sahip saat çevrimi sinyalleri oluşturulup modüllere verilmiştir. Şekil 5.13.'te oluşturulan simülasyon ortamının mimarisi, Şekil 5.14.'te ise AMD Xilinx Vivado platformundaki hiyerarşi gösterilmiştir.




**Şekil 5.13.** Simülasyon Ortamı Mimarisi




**Şekil 5.14.** Simülasyon Ortamı Hiyerarşisi

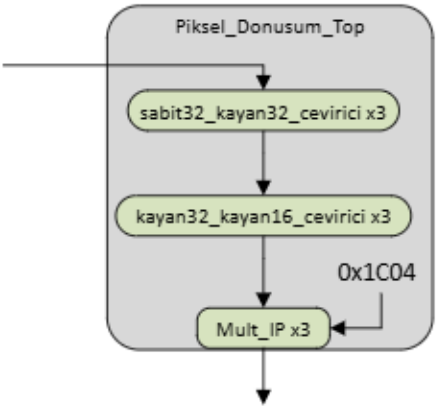
### 5.3.4. Modelin FPGA'da Gerçeklenmesi ve Simülasyonu

YOLOv11n modelinin ağırlık parametrelerinin kapladığı hafıza birimi yaklaşık olarak 2 MB civarındadır. Bu boyutta bir hafıza kapasitesinin FPGA içerisinde kullanılması evrişim operasyonu esnasında kullanılacak FIFO'ların sayısının limitlenmesine sebep olmaktadır. Bu sebeple, YOLOv11n evrişim parametrelerinin QSPI Flaş entegresi içerisine kaydedilip, bir ilklendirme sekansı ile bu parametrelerin başlangıçtaki evrişim operasyonları için okunarak, kalanının da çalışma zamanında okunarak idame edilmesine karar verilmiştir. QSPI Flaş entegresi, her AMD Xilinx Kintex-7 serisinin önyükleme yapabilmesi adına operasyonel kodunu barındıran ve yüksek miktarda alana sahip olan hafıza birimidir. Bu

nedenle evrişim ağırlıklarını saklayabilecek kadar bir alan mevcuttur. Evrişim ağırlıklarının aksine önyargı parametreler, 7560 adettir ve FPGA içerisinde saklanmaya elverişlidir. Bu yüzden önyargı değerleri bir ".coe" uzantılı dosya ile ROM içerisine kaydedilmiştir.

FPGA ortamında işlenecek video çerçevelerine bir ön işleme aşaması gerekmektedir. Ön işleme aşamasında yapılması gereken operasyon ise sisteme ulaşan 1024x768 boyutlarındaki video çerçevelerinin 640x640 boyutuna dönüştürülmesi gerekmektedir. Bu dönüşüm süreci, mantıksal işlem karmaşıklığı ve piksel kaybı gibi sonuçlara yol açabileceğinden ötürü bu aşamayı gerçekleştirmek adına 1024x768 boyutundaki video çerçevesinin ortasındaki 640x640 boyutunda alanı filtreleyen bir maskeleme modülü oluşturulmuştur. Maskeleme modülü 65MHz frekansına sahip bir saat çevrimi ile çalışmaktadır.

Dışarıdan gelen video verisinde her seferinde 24-bitlik bir vektör elde edilmektedir. Bu vektör RGB katmanlarındaki 8-bitlik, 0-255 değer aralığı bulunan pikselleri içermektedir ve FPGA içerisinde işlemler 16-bit (half) kayan noktalı gösterim formatında gerçekleşmektedir. Bundan dolayı maskelenen video çerçevesindeki piksellerin 16-bit kayan noktalı gösterim formatına dönüştürülerek 0-1 arasına normalize edilmesi gerekmektedir. Dönüşüm işleminin gerçekleştirilebilmesi için öncelikle 8-bitlik 3 kanala ayrılmış olan piksel değerleri 32-bit'e genişletilerek 32-bit sabit noktalı gösterimden 32-bit kayan noktalı gösterim formatına dönüştürülmüştür. Ardından 32-bit kayan noktalı gösterim formatından 16-bit kayan noktalı gösterim formatına dönüştürülmüştür. Veri tipi değiştirilmiş olan piksel değerleri 0-1 aralığında normalize edilmesi için $\frac{1}{255}$'in (0,0039) 16-bit kayan noktalı gösterim değeri olan 0x1C04 sayısı ile çarpılmıştır. Piksel dönüşüm modülü 65MHz frekansa sahip bir saat çevrimi ile çalışmaktadır. Şekil 5.15.'te bu operasyonların gerçekleştirilmesi için oluşturulan "Piksel_Donusum_Top" isimli modülün mimarisi görülebilmektedir.



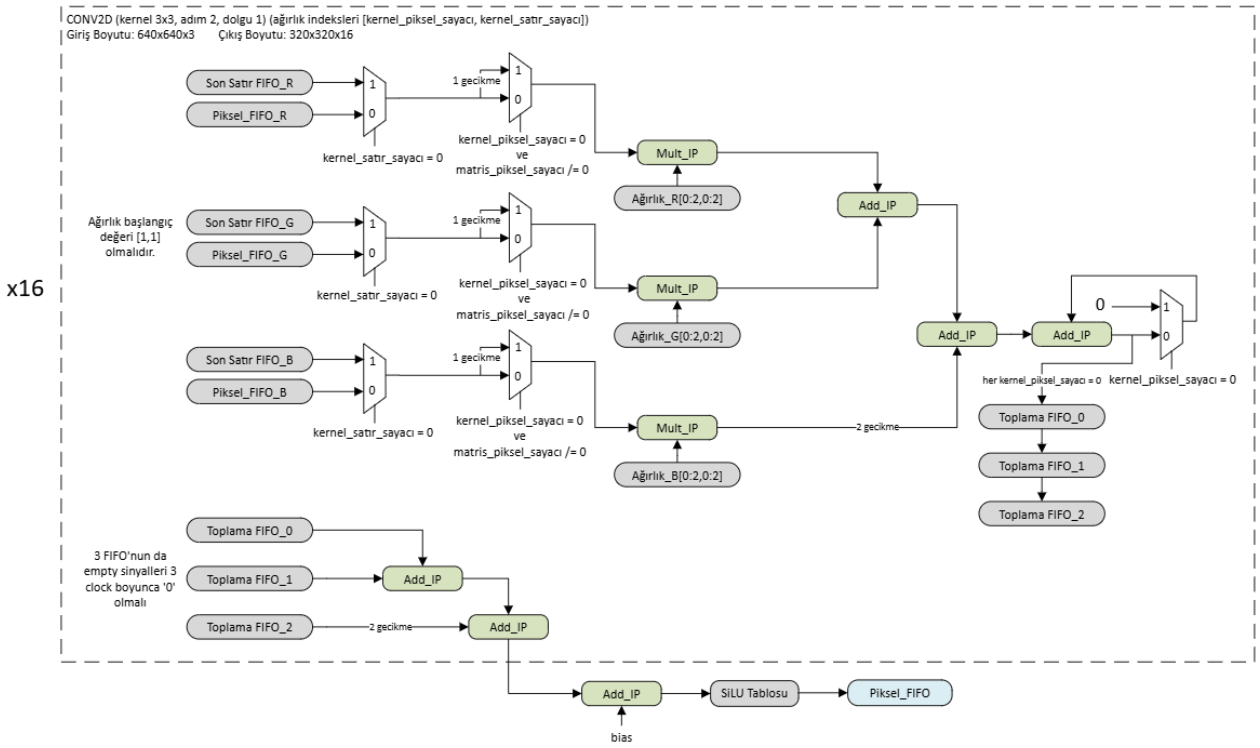

**Şekil 5.15.** Piksel Dönüşüm Modülü Mimarisi

Piksel dönüşümünü yapan modülün çıkışından alınan değerler ise 65MHz saat çevrimi alanından 200MHz saat çevrimi alanına geçiş yapılabilmesi, yakalanan pikselin sisteme kaçırılmadan verilmesi ve pikselin yakalandığının geri dönütünü FIFO boş sinyalini takip edilebilmesi adına her bir renk kanalına tahsis edilmiş toplam 3 adet FIFO'ya yazılmaktadır.

Ağırlıkların ve RGB piksellerin tutulduğu FIFO'lara ek olarak 4 adet daha FIFO grubu bulunmaktadır. Bu gruplardan ilki birleştirme (concat) operasyonunun yapılması için işlenen piksellerin sonuçlarını tutan Concat_FIFO grubudur. İkinci grup ise evrişim operasyonunda atılan adım miktarı dolayısıyla son satır ve önceki satırın yeniden işleme tabi tutulması adına bu satırları tutan Son_ve_Onceki_Satir_FIFO grubudur. Üçüncü grup, ağırlıklarla çarpılan piksel değerlerinin kanal bazlı kendi aralarında toplanma sonuçlarını ve maksimum havuzlama katmanlarının satır bazlı karşılaştırma sonuçlarını tutan Toplama_Karsilastirma_FIFO grubudur. Son grup ise evrişim operasyon sonuçlarının bir sonraki evrişim katmanına aktarılması için işlenmiş piksel değerlerini tutan Piksel_FIFO_RAM grubudur.

Yukarıda bahsedilen neredeyse bütün modüllerin bağlandığı ve YOLOv11n modelinin idamesini, noktalı gösterim IP çekirdeklerinin giriş-çıkış yönetimini, müsaitlik idaresini, FIFO yazma/okuma işlemlerini ve ağırlıkların yönlendirilmesi operasyonlarını gerçekleştiren son modül olarak YOLOv11_Kontrolcu modülü bulunmaktadır. Bu modül içerisinde çarpma ve toplama işlemlerini gerçekleştiren IP çekirdekler bulunmaktadır. Aynı zamanda SiLU, Sigmoid fonksiyonları ve sabit sayı vektörleri için gereken LUT'lar yer almaktadır.

YOLOv11_Kontrolcu modülünün yönettiği 8 adet ana operasyon vardır. Bu operasyonlardan ilki, 2 adımlı 1 dolgulu 3x3 evrişim operasyonudur. Şekil 5.16.'da evrişim katmanlarından ilki olan ve 2 adımlı 1 dolgulu 3x3 evrişim operasyonlarından birisi olan ilk evrişim katmanının (Kernel boyutu 3x3x3, tekrar sayısı 16) FPGA mimarisi gösterilmiştir. Diğer 2 adımlı 1 dolgulu 3x3 evrişim katmanlarının hepsinin ana yapısı aynı olup sadece giriş matrisinin içerdiği kanal sayısına göre kayan noktalı IP çekirdek kullanımı değişiklik göstermektedir.



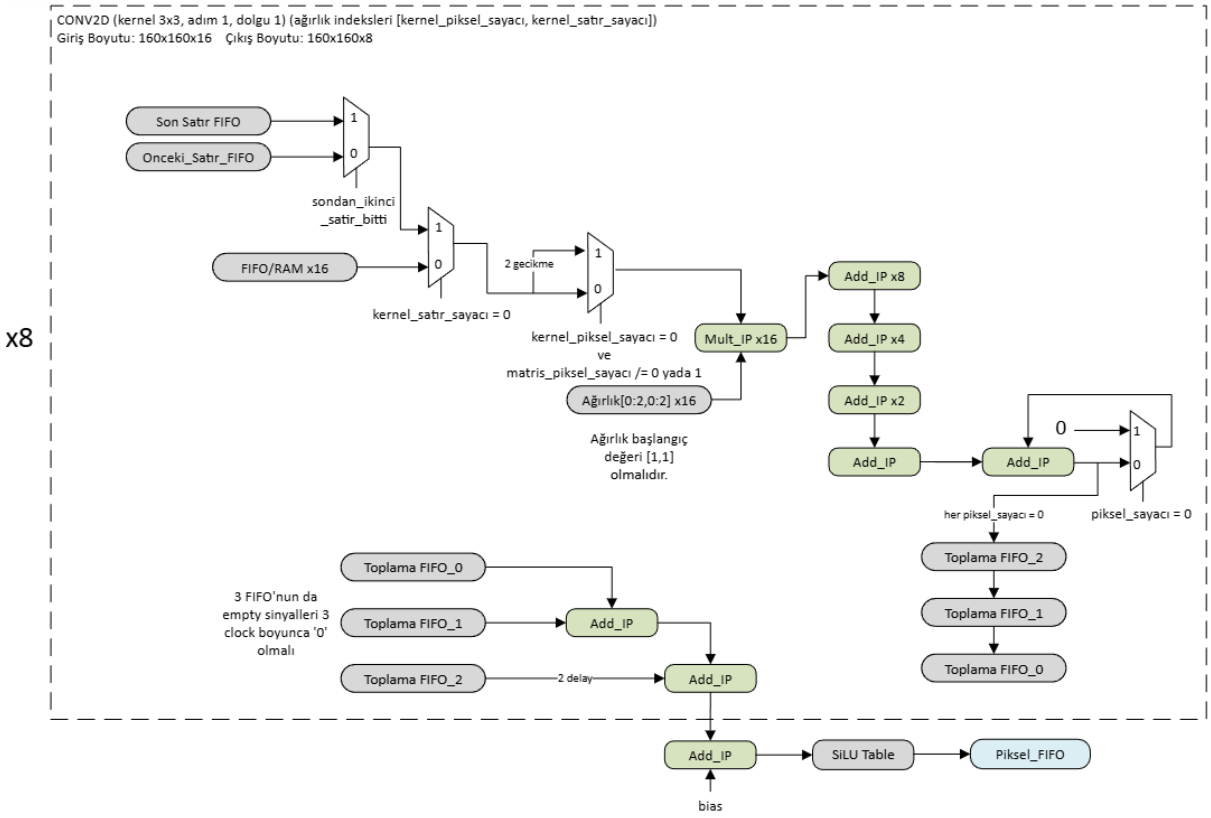

**Şekil 5.16.** 2 Adımlı 1 Dolgulu 3x3 Evrişim Operasyonu Mimarisi

Evrişim operasyonunda dolgu sayısı 1 olduğundan dolayı ağırlık başlangıç değerleri [1,1] olmaktadır. Bu sayede, sanki dolgu varmış gibi işlem yapılabilir ve FIFO'lara önden sıfır doldurma gereksinimi ortadan kalkar. Ana operasyon şu şekildedir; önce girdi olarak verilen matrisin her kanalındaki pikseller kendi ağırlıkları ile çarpılır. Ardından bu ayrı kanallara ait pikseller kendi aralarında toplanır. Böylece kernelin bütün kanalları toplanmış olur. Sonrasında aynı kernelin aynı satırında bulunan piksel değerleri için de aynı operasyon gerçekleştirilir. Kernelin aynı satırında bulunan pikseller de kendi aralarında toplanır ve sonuç Toplama FIFO adı verilen FIFO'lara yazılır. FIFO'lara yazılmasının sebebi ise aynı işlemin kernel satırları için de yapılacak olmasıdır. En nihayetinde kernelin bütün kanalları, satır ve sütunları birbirleriyle toplanarak ana evrişim operasyonu tamamlanmış olur. Toplama FIFO'larından alınan toplam sonuçları en son ön yargı (bias) ile toplanır ve SiLU

LUT'a giriş olarak verilir. LUT çıkışı işlenmiş verinin bir sonraki adıma geçirilebilmesi adına piksel FIFO'suna ya da RAM'ine yazılır.

Giriş piksellerinin bir önceki piksel seçilmesi ise adım sayısından kaynaklanmaktadır. Kernel kaydırılırken bir önceki adımdaki son sütun bir sonraki adımın ilk sütunu olacağından dolayı son sütundaki pikseller tutulur ve bir adım bittikten sonrasında sonraki adımın ilk girdisi olarak sisteme yeniden verilir. Bu işlemler yapılırken video saat çevriminin 65 MHz olması ve sistemin 200MHz ile çalışmasından dolayı herhangi bir piksel kaçırılmaz veya FIFO'lar dolmadan bir sonraki piksel işleme alınabilir. Aynı durum satırlar için de geçerlidir. Kernel bir alt satıra geçtiği zaman işlem yapılan en son satırdaki piksellerin hepsi yeniden işleme tabi tutulmalıdır. Bu sebeple kernel_satır_sayacı değeri 2 olduğu zaman gelen çerçeve satırı bir başka FIFO'da tutulur ve alt satıra geçildiği zaman önce doldurulan FIFO işleme alınır.

İkinci operasyon, 1 adımlı 1 dolgulu 3x3 evrişim operasyonudur. Şekil 5.17.'de 1 adımlı 1 dolgulu 3x3 evrişim katmanlarından bir tanesinin (Kernel boyutu 3x3x16, tekrar sayısı 8) FPGA mimarisi gösterilmiştir. Diğer 1 adımlı 1 dolgulu 3x3 evrişim katmanlarının hepsinin ana yapısı aynıdır fakat girişin içerdiği kanal sayısına göre kayan noktalı IP çekirdek kullanım sayısı farklıdır.

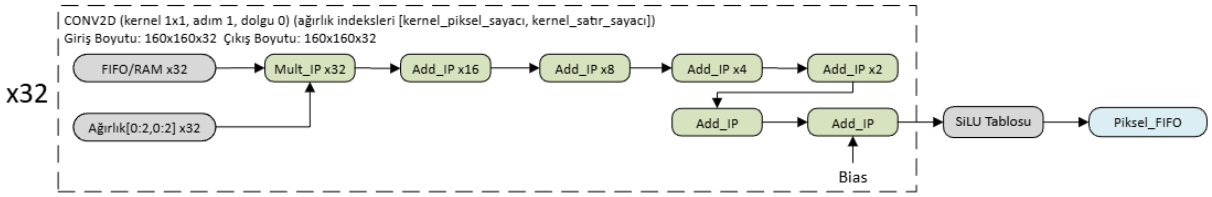



**Şekil 5.17.** 1 Adımlı 1 Dolgulu 3x3 Evrişim Operasyonu Mimarisi

Bu evrişim katmanında gerçekleştirilen ana operasyon 2 adımlı 1 dolgulu 3x3 evrişim katmanındakiler ile birebir aynıdır. İki operasyon arasındaki fark adım sayısından kaynaklı giriş piksellerinin seçilme mantığıdır. 2 adımlıda kernelin son pikselinin veya son satırın yeniden işleme alınması gereklidir. 1 adımlı olan evrişim operasyonunda ise bu durum son iki piksel veya son iki satır olarak karşımıza çıkmaktadır. Giriş piksellerinin FIFO'ya yazılış periyodundan dolayı bir sonraki piksel gelene kadar iki piksel işlemek veya iki satır işlemek piksel kaybına sebep olmamaktadır. Bir kernel tamamlandıktan sonra sondan iki önceki piksel ya da satır yeniden işleme çarpılır ve diğer sonuçlar ile toplanır. İşleme yeniden alma operasyonu pikselin 2 kez geciktirilmesi ile mümkündür. Bir sonraki adımda bir kez geciktirilen piksel yeniden geciktirileceğinden dolayı sisteme yeniden iki kez geciktirilmiş piksel verilmelidir. Bu iki operasyon yapıldıktan sonrasında yeni gelen piksel kernelin son adımı olarak operasyona dahil edilmektedir.

Modül tarafından yönetilen üçüncü operasyon 1 adımlı 1x1 evrişim operasyonudur. Şekil 5.18.'de 1 adımlı 1x1 evrişim katmanlarından bir tanesinin (Kernel boyutu 1x1x32, tekrar sayısı 32) FPGA mimarisi gösterilmiştir. Diğer 1 adımlı 1x1 evrişim katmanlarının

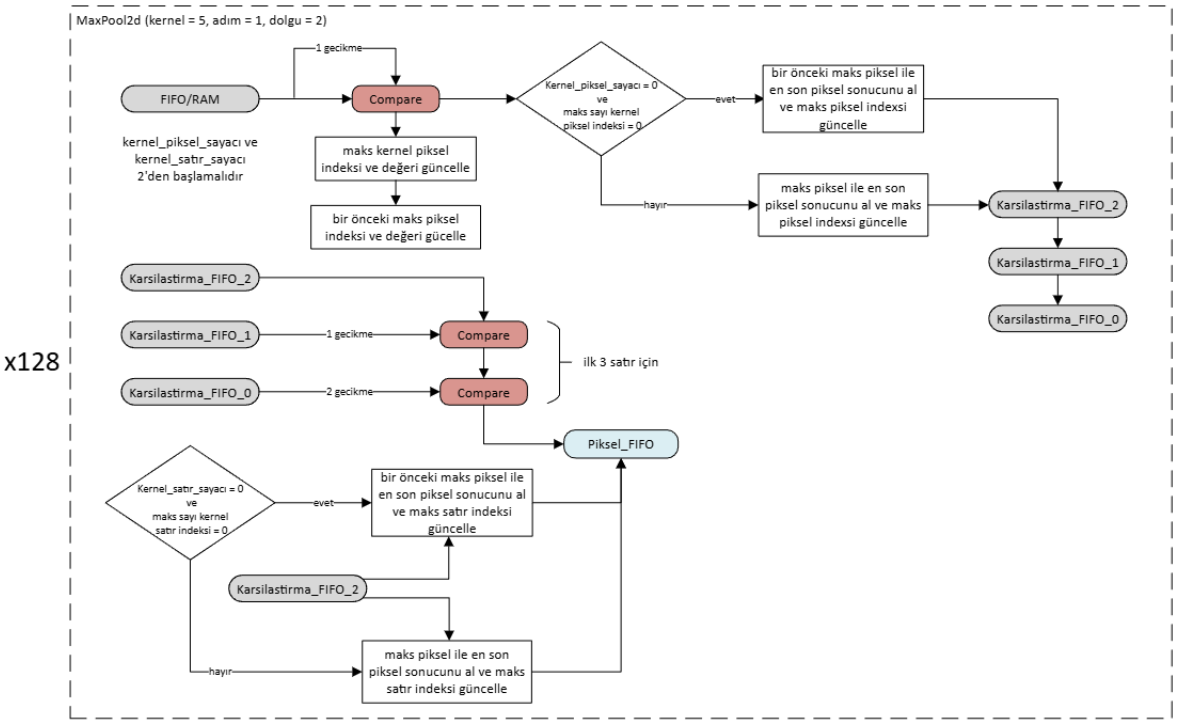
hepsinin ana yapısı aynıdır fakat girişin içerdiği kanal sayısına göre kayan noktalı IP çekirdek kullanım sayısı farklıdır.




**Şekil 5.18.** 1 Adımlı 1x1 Evrişim Operasyonu Mimarisi

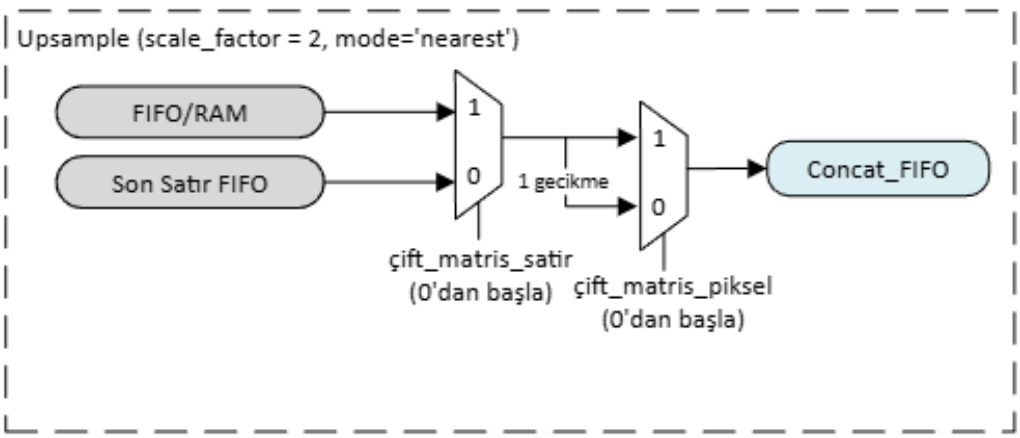
1 adımlı 1x1 evrişim katmanında işlemler daha basittir. Katmana gelen pikseller çarpılmaları gereken ağırlıklar ile çarpılarak kendi aralarında toplanmaktadır. Toplama sonucuna önyargı değeri de eklenerek SiLU LUT'a gönderilmektedir. LUT sonucu FIFO'ya yazılmaktadır.

Diğer iki operasyon ise 5x5 maksimum havuzlama ve üst örnekleme (upsample) operasyonlarıdır. Şekil 5.19.'da 5x5 maksimum havuzlama katmanlarından bir tanesinin, Şekil 5.20.'de ise üst örnekleme katmanının FPGA mimarisi gösterilmiştir. Diğer maksimum havuzlama ve üst örnekleme katmanları da aynı mimariye sahiptir.




**Şekil 5.19.** 5x5 Maksimum Havuzlama Mimarisi

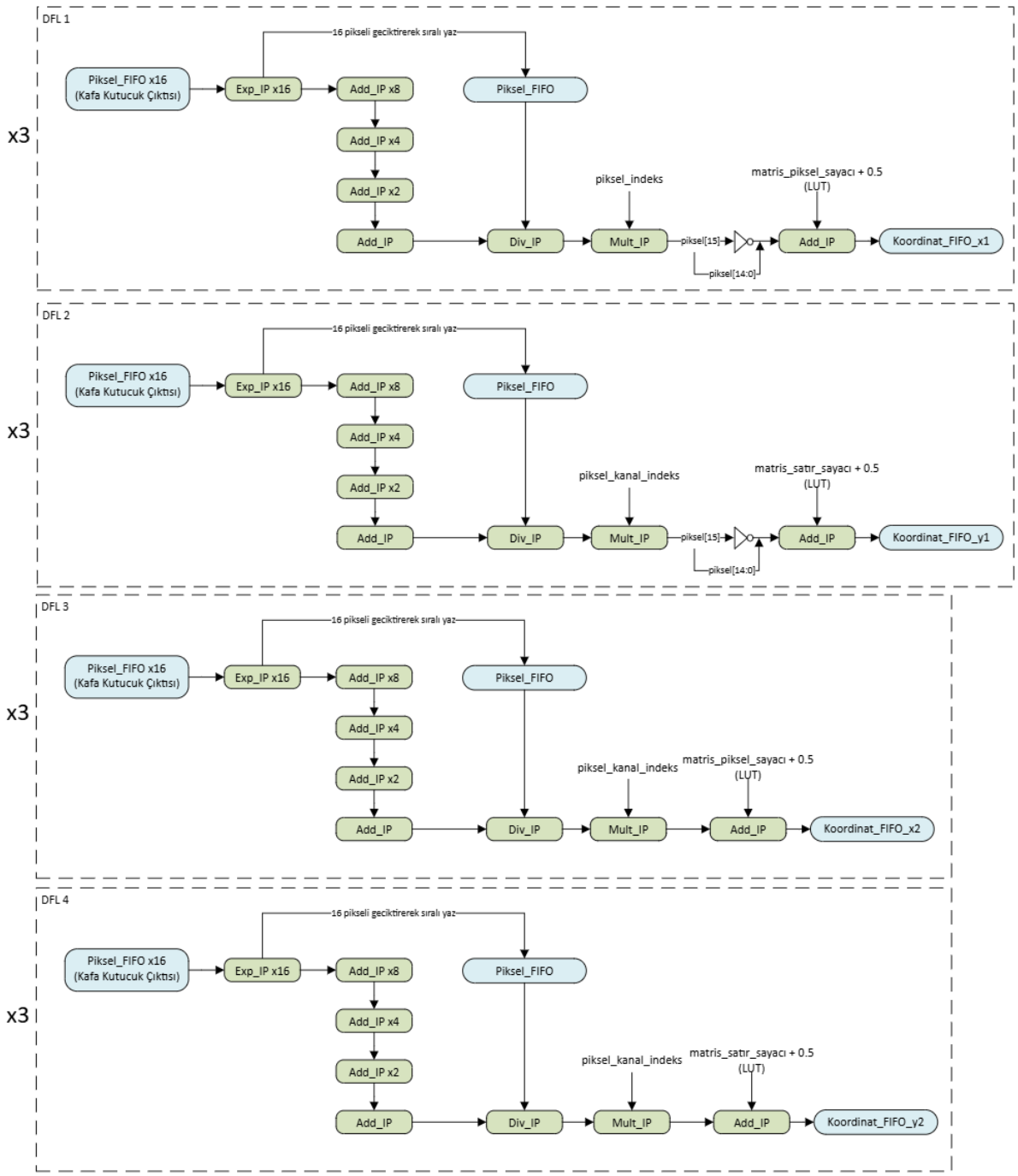
Maksimum havuzlama operasyonunda dolgu değeri 2 olduğu için kernel sayaçları 2'den başlamaktadır. Sisteme verilen bir piksel değeri aynı kernelde en son kayıtlı olan maksimum sayı ile kıyaslanır ve maksimum sayının kernel indeksi ve değeri güncellenmektedir. Aynı zamanda bir sonraki adım için maksimum ikinci sayı da güncellenmektedir. Bir sonraki adımda ise maksimum sayının kernel indeksi sıfır ise en büyük ikinci sayı ile yeni gelen piksel kıyaslanarak yeniden en büyük sayı ve en büyük ikinci sayı ve indeksleri güncellenir. Aynı durum satırlar arası kıyaslama için de geçerlidir. Sadece ilk satırın kıyaslanması esnasında 3 adet FIFO kullanılmaktadır. Ondan sonraki kıyaslamada sadece yeni gelen satır kullanılacağından dolayı yalnızca 1 adet FIFO kullanılmaktadır.




**Şekil 5.20.** Üst Örnekleme (Up Sample) Mimarisi

Üst örneklemede amaç her bir pikselin 2x2'lik bir hale çevirerek matrisi yeniden oluşturmaktır. Bu yüzden bir satır oluşturulurken piksel değeri hem satıra hem sütuna iki kez yazılır. Bu da pikseli 1 adım geciktirerek ve son gelen satırı tutarak bir kez daha o satırı yeni FIFO'ya yazarak gerçekleştirilmektedir.

Operasyonlardan bir diğeri olan DFL (Distribution Focal Loss – Dağıtım Odak Kaybı) operasyonları aynı mimariye sahip 4 farklı operasyondur ve her birisinin çıktısı görüntüde tespit edilen kusurların sınırlayıcı kutularını göstermektedir. DFL operasyonu sonucunda kusurun tespit edildiği sınırlayıcı kutuların değerleri ve bu değerlerin köşe noktalarının belirlenmesi için hazır bir formata bürünmüştür. Şekil 5.21.'de DFL operasyonlarının FPGA içerisindeki mimarisi gösterilmiştir.

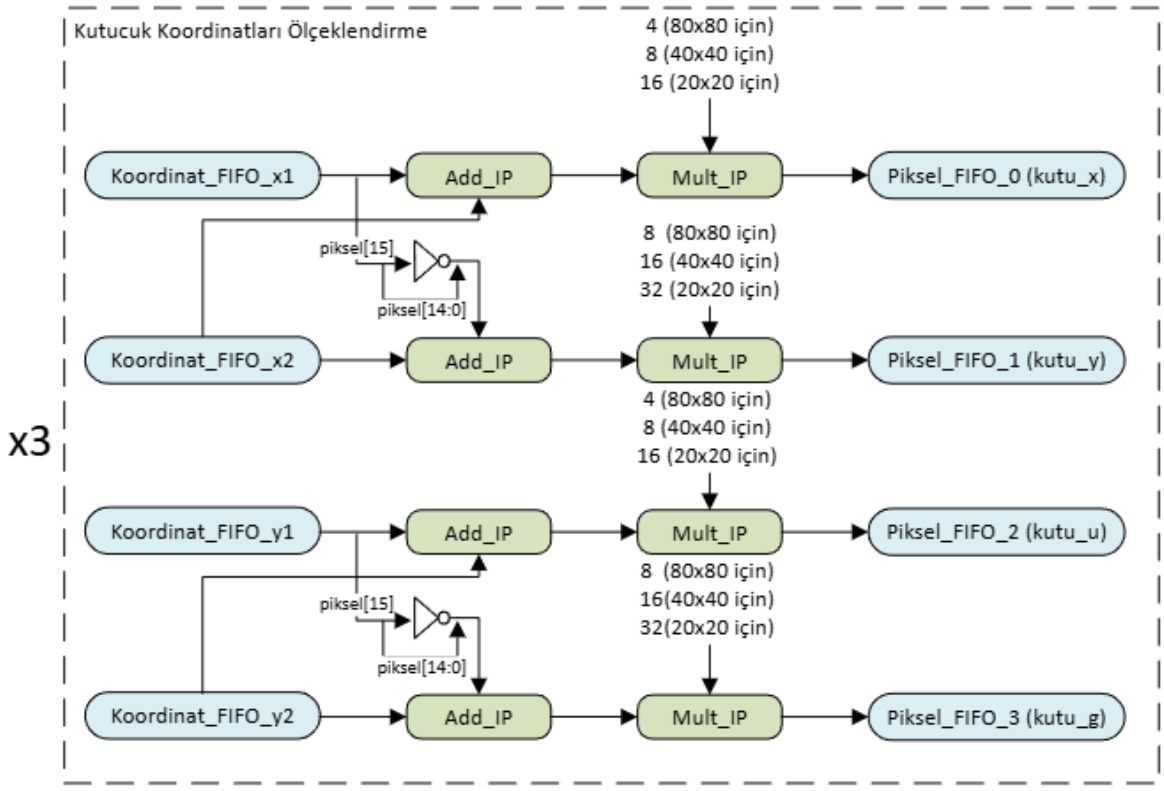



**Şekil 5.21.** DFL ile Koordinatların Belirlenme Mimarisi

Kafa katmanı çıktılarından sınırlayıcı kutu bilgilerini taşıyan 64 katmanlı çıktıyı koordinat bilgisine dönüştürmek amacıyla öncelikle kafa çıktısı 16 katmanlı 4 ana gruba ayrılmaktadır. Bu gruplar x1, x2, y1 ve y2 değerlerini belirleyecek olan gruplardır. Her bir kafa çıktısı için her katmanın elementlerinin dağılımını normalize etmek adına Softmax fonksiyonuna sokulur. Bundan dolayı üstel IP çekirdeği kullanılmıştır. Her bir element toplama bölüneceği için 16 element sıralı şekilde bir FIFO'ya yazılmıştır ve toplam elde edildiği zaman elementler teker teker toplam değerine bölünmüştür. Ardından element

kaçıncı kanalda ise o kanal sayısıyla (başlangıç değeri 1) çarpılmıştır. Son işlem olarak x1 ve y1 koordinatları için element kaçıncı satırda ise (başlangıç değeri 0) element o satır değerinin 0,5 fazlasından çıkartılmış, x2 ve y2 koordinatları için ise bu değer ile toplanmıştır.

Son iki operasyon ise sınırlayıcı kutuların koordinatlarını ölçeklendirme ve sınıf skorlarını düzenleyen operasyonlardır. Bu iki operasyon sonucunda elde edilen değerler artık tespit edilen sınıfların skorlarını ve sınırlayıcı kutuların x-y koordinat bilgileri, uzunluk ve genişlik değerlerini içermektedir. Şekil 5.22.'de kutucuk koordinatlarını ölçeklendirme yapısının, Şekil 5.23.'te ise sınıf skorları belirleme yapısının FPGA içerisindeki mimarisi gösterilmiştir.




**Şekil 5.22.** Kutucuk Koordinatlarını Ölçeklendirme

Koordinatlar belirlendikten sonrasında koordinatların ölçeklendirilmesi gerekmektedir. Ölçeklendirme işleminde kullanılan formüller x değeri x koordinatı, y değeri y koordinatı, u değeri uzunluk ve g değeri genişlik olacak şekilde x1, x2, y1 ve y2 cinsinden Eşitlik 5.4'te gösterilmiştir.

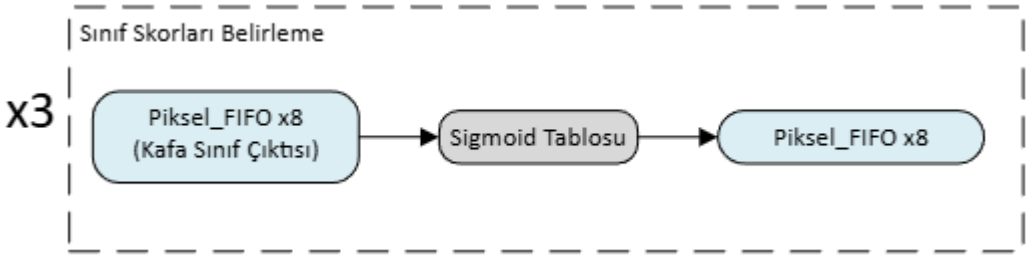
$$x = \left(\frac{x1+x2}{2}\right) \times Oran$$

$$y = \left(\frac{y1+y2}{2}\right) \times Oran$$

$$g = (x2 - x1) \times Oran \tag{5.4}$$

$$u = (y2 - y1) \times Oran$$

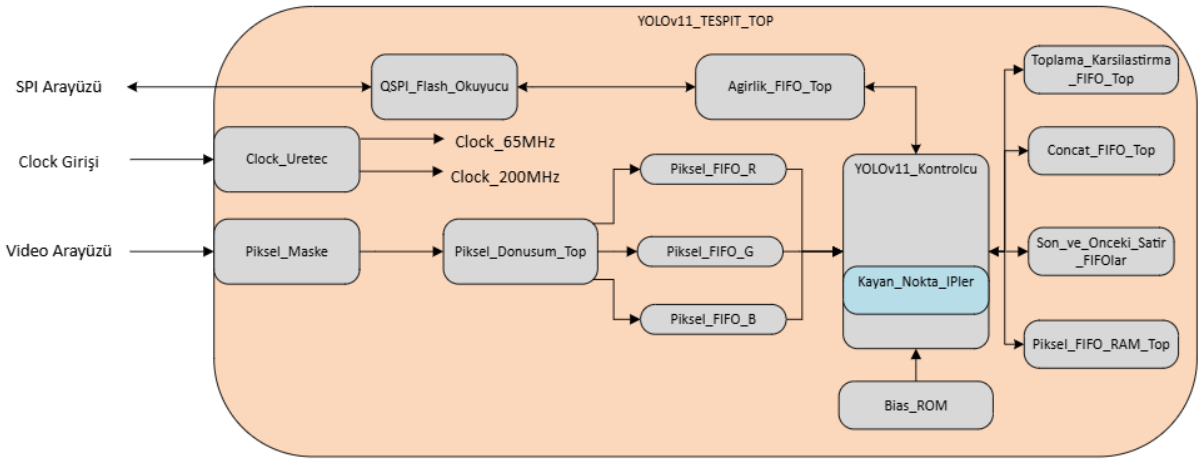
$$Oran = \begin{cases} 8, matris\ boyutu = 80 \times 80 \\ 16, matris\ boyutu = 40 \times 40 \\ 32, matris\ boyutu = 20 \times 20 \end{cases}$$



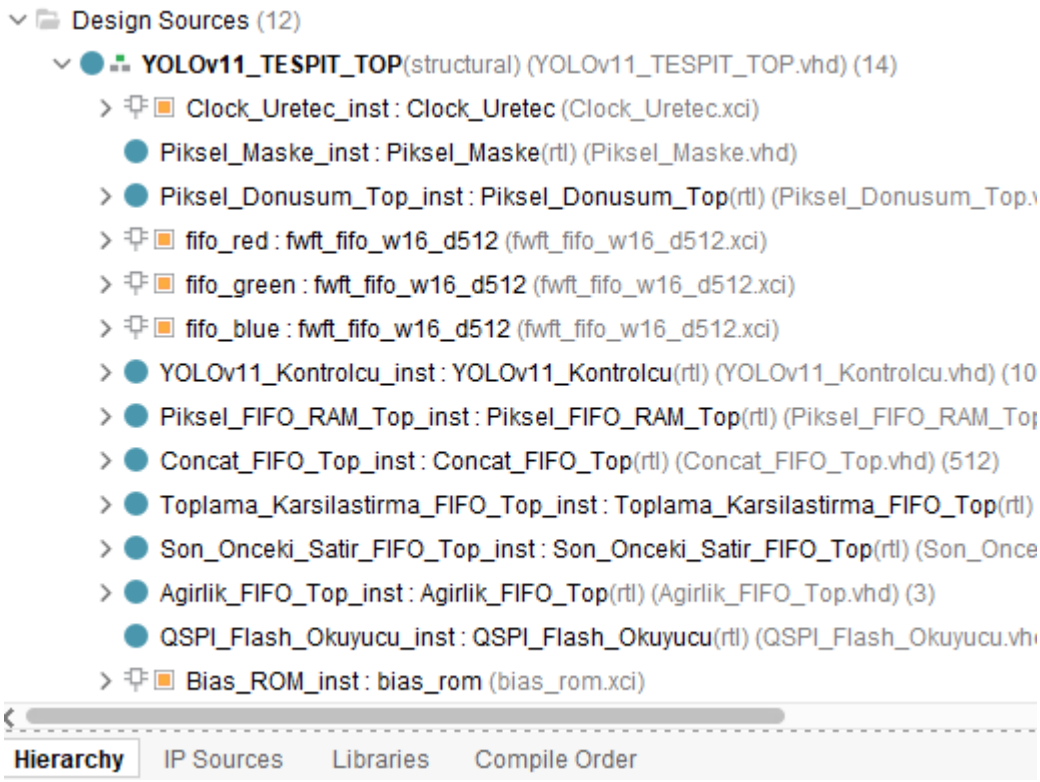

**Şekil 5.23.** Sınıf Skorlarını Belirleme Mimarisi

Kafa çıktılarından sınıf bilgisini taşıyan 8 katmanlık çıktıyı sınıf skorlarına çevirmek adına her bir element Sigmoid fonksiyonuna girdi olarak verilir ve alınan çıktı katmandaki sınıf değeri için bir güven skoru değeridir. Bu skor değeri ile yapılan tespitin hangi sınıfa ait olduğu belirlenmektedir.

Oluşturulan bütün mimariler YOLOv11_TESPIT_TOP isimli modülde toplanarak birleştirilmiştir ve aralarındaki iletişim sağlanmıştır. Dışarıdan alınacak bir saat çevrimi içeride 65MHz ve 200MHz frekanslarına dönüştürülerek gerekli modüllere bağlanmıştır. Şekil 5.24.'te oluşturulan çatı modülü ve içerdeki bağlantı mimarisi görülmektedir. Şekil 5.25.'te ise AMD Xilinx Vivado platformundaki hiyerarşi gösterilmiştir.




**Şekil 5.24.** Çatı Modülü Mimarisi




**Şekil 5.25.** Çatı Modül Hiyerarşisi

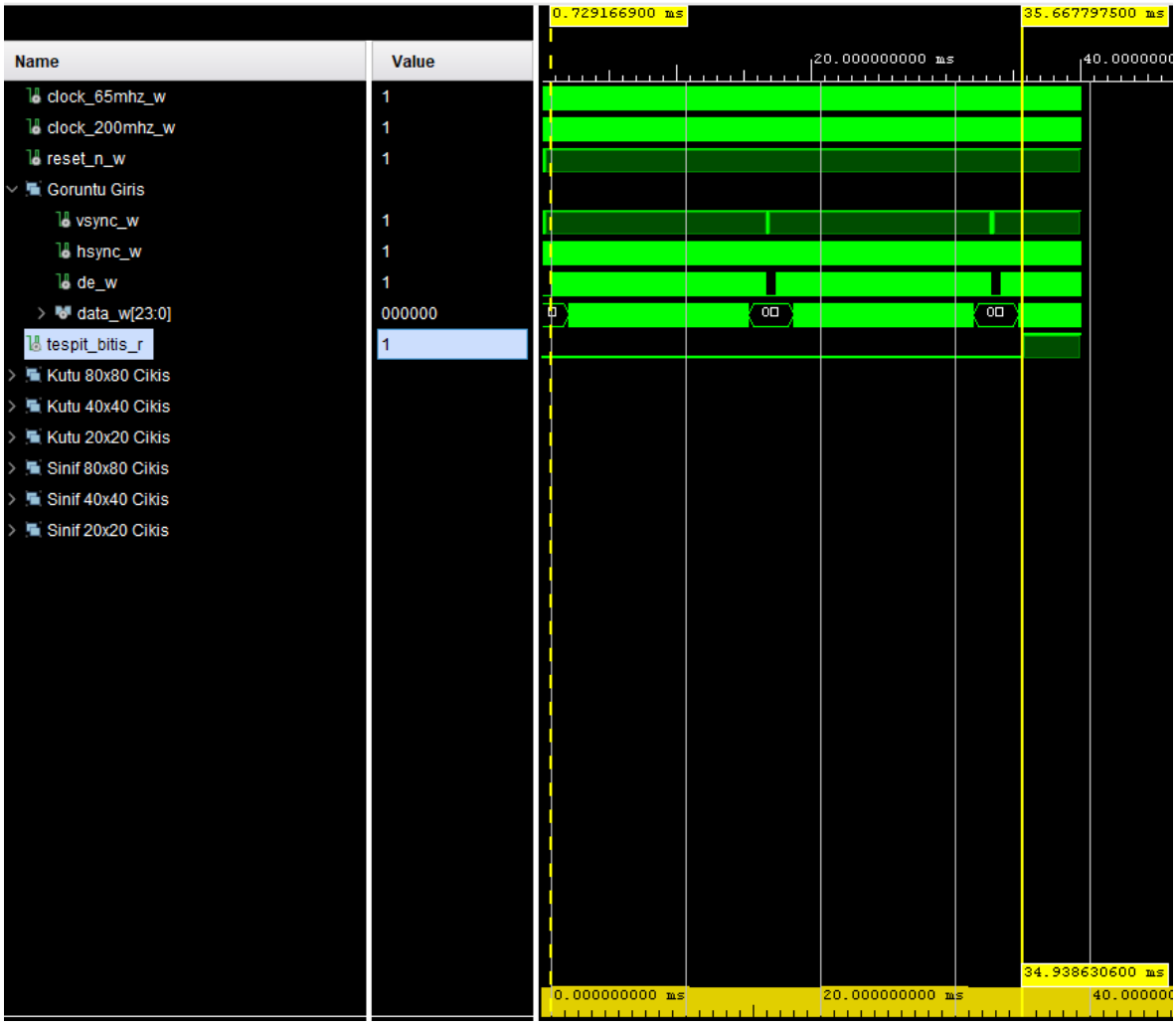
Bütün modüller tek çatı altında toplandıktan sonrasında sentezleme işlemi gerçekleştirilmiştir. Başarılı bir sentez sonucu alındıktan sonrasında mimarinin FPGA içerisindeki kaynak tüketimi incelenmiştir. Tablo 5.5.'te sentez sonucunda AMD XILINX Vivado platformu tarafından mimarinin FPGA'daki tahmini kaynak tüketimi tablosu verilmiştir.

**Tablo 5.5.** Sentez Sonrası FPGA Kaynak Tüketimi

| Kaynak | Tahmin | Müsait | Kullanım Yüzdesi (%) |
|---|---|---|---|
| LUT | 251300 | 254200 | 98,85 |
| FF | 412988 | 508400 | 81,23 |
| BRAM | 732 | 795 | 92,07 |
| DSP | 1344 | 1540 | 87,27 |
| BUFG | 2 | 32 | 6,25 |
| MMCM | 2 | 10 | 20 |

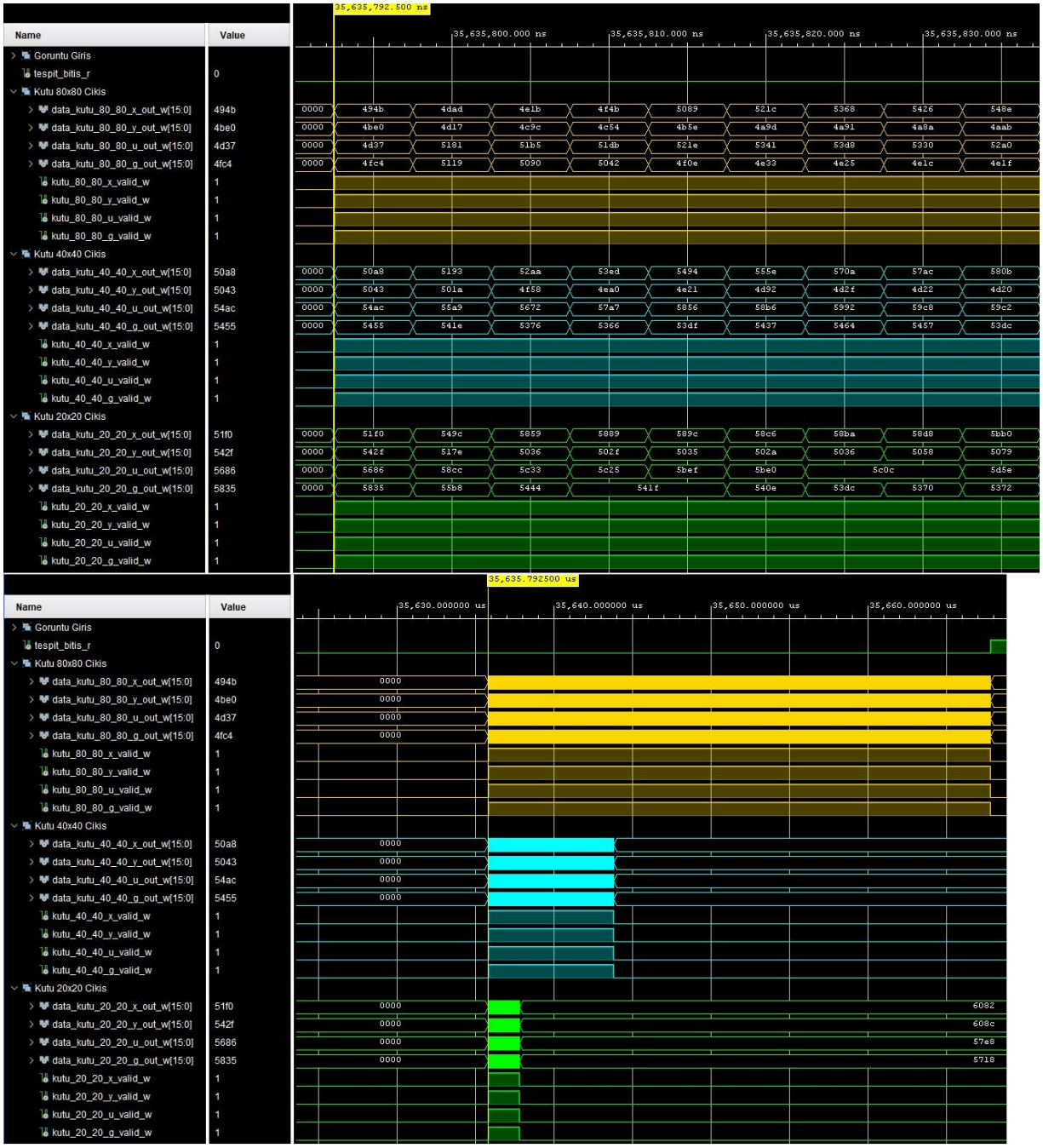
Oluşturulan FPGA mimarisi simülasyon ortamında test edilmiştir. FPGA'den alınan çıktılar bir ".txt" uzantılı dosyaya yazılarak kaydedilmiştir. Ardından bu dosya Python

ortamına okunarak birleştirilmiş ve sonraki işleme (post-processing) aşaması gerçekleştirilerek tespit sonuçları görselleştirilmiştir. Şekil 5.26.'da simülasyon ortamında görüntünün ilk pikseli geldikten sonra tahminin bitişine kadar geçen süre (tahmin süresi), Şekil 5.27.'de sınırlayıcı kutuların verilerinin çıkışları, Şekil 5.28.'de ise sınıf verilerinin çıkışları, Şekil 5.29.'da ise simülasyon ortamı çıktılarının sonraki işleme aşamasının gerçekleştirilerek görselleştirilmesi sonucunda elde edilen görüntülerden bazıları verilmiştir.



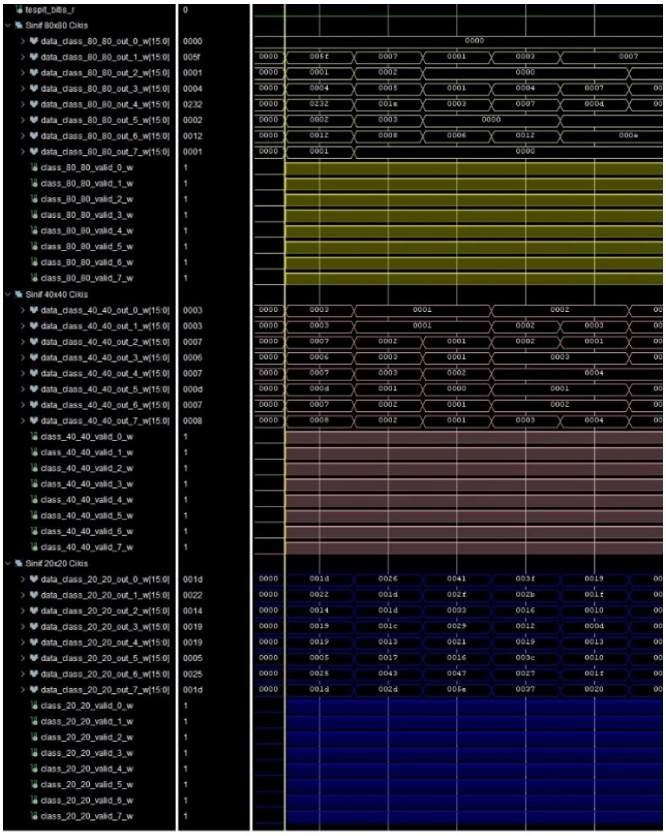

**Şekil 5.26.** Simülasyon Üzerinde Tespit Süresi

Simülasyonda görülen sonuçlara göre video çerçevesinin ilk pikseli sisteme ulaştıktan 34,9386306ms sonra çıkarım süreci tamamlanarak veriler sonraki süreç işlemelerine hazır olmaktadır. Bir diğer deyişle çıkarım süresi 34,9386306ms'dir fakat görüntünün ilk pikselinin geldiği andan itibaren modüller operasyonlarına başladıkları için ve bir çerçevenin tamamen sisteme ulaştırılma süresi 16,7ms olduğundan dolayı bir çerçeve geldikten sonra 18,2386306ms içerisinde tahmin işlemleri tamamlanmaktadır.




**Şekil 5.27.** Sınırlayıcı Kutu Değerleri Çıktıları

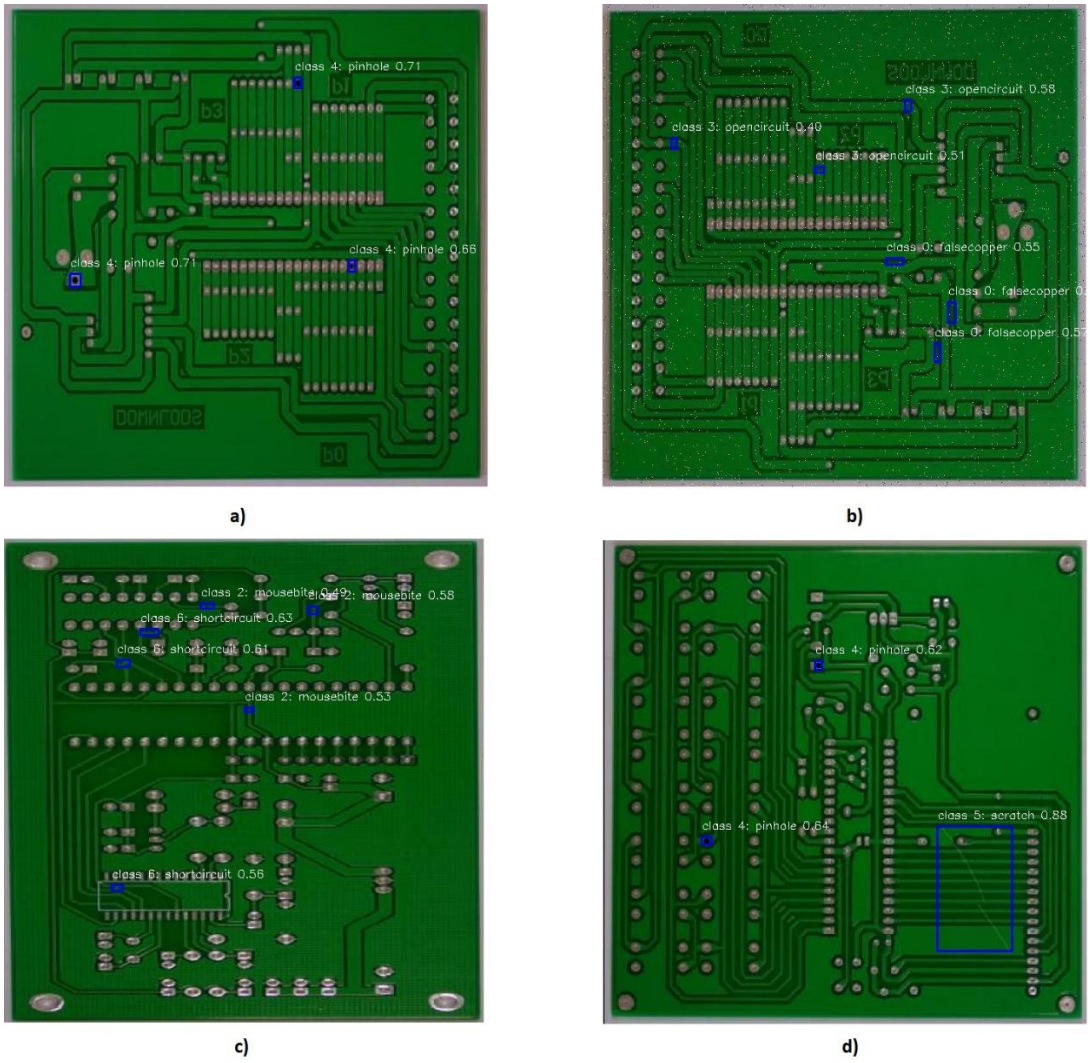
Sınırlayıcı kutu değerleri 80x80, 40x40 ve 20x20 matris boyutlarında 3 farklı çıktı grubuna sahiptir. Şekil 5.27.'de alınan çıktıların başlangıç zamanı ve değerleri on altılık taban formatı ile gösterilmiştir.




a)


b)

**Şekil 5.28.** a) Sınıf Çıktıları Başlangıcı ve Değerleri b) Bütün Sınıf Çıktıları Süreci




**Şekil 5.29.** FPGA Tahmin Çıktısının Görselleştirilmiş Hali; a) pinhole, b) opencircuit ve false copper, c) mousebite ve short circuit, d) pinhole ve scratch

Yapılan simülasyonlar sonucunda bütün test görüntüleri için çıktılar alınarak Python ortamında mAP50 cinsinden doğruluk hesabı gerçekleştirilmiştir. Doğruluk değerleri mAP50 cinsinden Falsecopper sınıfı için %98,9, Missinghole sınıfı için %98,8, Mousebite sınıfı için %97,5, Open Circuit sınıfı için %94,9, Pinhole sınıfı için %98,9, Scratch sınıfı için %99,4, Short Circuit sınıfı için %98,5 ve Spur sınıfı için %96 olarak hesaplanmıştır.

## 5.4. Sonuçların Karşılaştırılması

Alınan sonuçlar dahilinde önce sistemin 32-bit kayan noktalı gösterim ile çalıştırıldığında elde edilen hız ve doğruluk performansları, ardından literatürde yapılan diğer çalışmalarda elde edilen hız ve doğruluk performansları bu tez çalışmasında gerçekleştirilen modelin çıktıları ile kıyaslanmıştır. Tablo 5.6.'da Python ortamında YOLOv11n modelinin Tek Duyarlı Format kayan noktalı gösterim ile çalıştırılmasıyla elde edilen sonuçlar ile FPGA çıktıları doğruluk ve hız açısından karşılaştırması verilmiştir.

**Tablo 5.6.** FPGA ve Python Çıktıları Kıyaslaması

| **Sınıf** | **FPGA (%)** | **Python (CPU Üzerinden)** | **Python (GPU Üzerinden)** |
|---|---|---|---|
| falsecopper | 98,9 | 99,4 | 99,4 |
| missinghole | 98,8 | 98,9 | 98,9 |
| mousebite | 97,5 | 98 | 98 |
| opencircuit | 94,9 | 95,3 | 95,3 |
| pinhole | 98,9 | 98,8 | 98,8 |
| scratch | 99,4 | 99,5 | 99,5 |
| shortcircuit | 98,5 | 98,7 | 98,7 |
| spur | 96 | 96,3 | 96,3 |
| Ortalama | 97,86 | 98,1 | 98,1 |
| **Hız** | **(ms)** | **(ms)** | **(ms)** |
| Ön İşleme | 0,00005 | 4 | 3 |
| Çıkarım | 34,93863 | 150,1 | 34,030199 |
| Sonraki İşleme | N/A | 1 | 1 |

Yapılan karşılaştırma sonucunda 32-bit kayan noktalı gösterim ve 16-bit kayan noktalı gösterim ve farklı platformlarda çalıştırılan YOLOv11n modeli için "pinhole" sınıfında daha iyi tahmin sonuçları elde edildiği görülmektedir. Genel başarı olarak ise 16-bit kayan noktalı gösterimlerde ondalıklı kısmın genişliğinin kısıtlanmasından dolayı performans kaybı meydana gelmektedir.

Literatürde daha önceden bu konuda yapılmış çalışmalar ile bu tez çalışmasında elde edilen sonuçların karşılaştırılması doğruluk başarısı en yüksekten en düşüğe doğru sıralanarak Tablo 5.7.'de verilmiştir. Karşılaştırma yapılan literatür çalışmaları aynı veri setleri ve birebir aynı cihazlar kullanılarak gerçekleştirilmemiştir. Bu farklardan dolayı başarı ve hız kıyaslaması birebir yapılamamaktadır. Ancak, genel baskılı devre kusur tespiti alanında yapılan farklı çalışmaların genel kıyaslaması olarak kullanılabilir. Bu tez çalışmasında kullanılan veri seti üzerinde FPGA tabanlı bir çalışma ilk defa gerçekleştirilmiştir.

**Tablo 5.7.** Literatürde Alınan Sonuçlar ile Performans Kıyaslaması

| **Çalışma (Referans Numarası)** | **Veri Seti** | **Kullanılan Cihaz** | **Kullanılan Model** | **Doğruluk** | **Hız** |
|---|---|---|---|---|---|
| [37] | Özel Oluşturulmuş | NVIDIA GeForce RTX 3080 | YOLOv4-MN3 | %98,64 | 56,98 FPS (17,55ms) |
| [31] | DeepPCB [31] | Titan X | GPP | %98,6 | 62 FPS (16,13ms) |
| [47] | DeepPCB [31] | AMD Xilinx Kria KV260 | YOLOv8m | %98,21 | 29432,9ms |
| [38] | HRIPCB [39] | Tesla T4 Graphics | Modifiye Edilmiş YOLOv5 | %98,1 | - |
| [25] | PCB-DATASET [26] | NVIDIA GeForce RTX 2070 | Skip Bağlantılı Evrişimsel Oto Enkoder | %98 | 55 FPS (18,18ms) |
| **Tez Çalışması** | **detection-pcb-defects-yolov8 [59]** | **AMD Xilinx Kintex-7 (XC7K410T-FFG900)** | **YOLOv11n** | **%97,86** | **34.94ms** |
| [33] | PCBA-DET [33] | NVIDIA GeForce RTX3080 | YOLOv5s | %97,3 | 322,6 FPS (3,1ms) |
| [21] | Halka Açık Veri Seti | AMD Xilinx PYNQ-Z2 | YOLOx-Plus | %90,1 | 72,6FPS (13,77ms) |
| [5] | DeepPCB [31] | XCZU5EV-2SFVC784I | YOLOv3 | %78,9 | 97,59ms |
| [44] | TDD-PCB [36] | NVIDIA GeForce RTX3070 | YOLOv5 + FPN | %77,53 | 90 FPS (11,11ms) |
| [40] | DeepPCB [31] | Google Colab | YOLOv4 | %74,62 | - |

# 6. SONUÇ VE ÖNERİLER

Derin öğrenme ile nesne tespiti, geleneksel yöntemlere nazaran daha avantajlı bir konuma sahiptir. Derin öğrenme aracılığı ile yapılan nesne tespitlerinde modelin boyutu, karmaşıklığı ve modelin entegrasyonunun yapıldığı donanımın yeterliliği ise modelin tespit hızını etkileyen başlıca unsurlardandır. Yeterli hıza ulaşmak için bu unsurların belirli bir seviyede olması gerekmektedir.

Gerçek zamanlı nesne tespiti için yeterli bir hıza sahip olması bir tartışma konusu olmasının yanı sıra YOLOv11n modeli baskılı devre kusurlarının tespiti açısından yeterli kapasiteye sahip bir modeldir. Hem eğitim hem de uygulama açısından modelin mAP50 değerleri %95'in üzerinde bir performans sergilemiş ve %98,4'lük bir eğitim başarısı yakalamıştır. 100 devir sayısında aynı parametreler ile eğitildiği zaman YOLOv8n modelinden %2,9 daha yüksek mAP50 başarısı elde etmiş, YOLOv8s modelinden ise %0,4 daha düşük mAP50 başarısı yakalamıştır. YOLOv8s ile arasında bulunan bu farka rağmen içeriğinde bulunan parametre sayısı YOLOv8s modelinin yaklaşık %20'si kadardır. Bundan dolayı YOLOv11n modeli, yüksek başarısı ve içerdiği parametre sayısı ile entegre edilebilirliği daha yüksek bir model olduğunu göstermektedir.

Bu tez çalışmasında, Kintex-7 (XC7K410T-FFG900) serisi bir FPGA'ya YOLOv11n modeli RTL tabanlı olarak gerçeklenip gerçek dünya girdileri ile aynı zamanlamaya sahip bir görüntü altyapısı oluşturularak simülasyon performansları gözlemlenmiştir. Ayrıca YOLOv11n modeli için 16-bit kayan noktalı gösterim formatı ile çalışan bir FPGA mimarisi sunularak önerilen mimarinin içeriği ve aşamaları anlatılmıştır. 32-bit kayan noktalı gösterim formatıyla Python ortamında çalıştırılan modelin genel doğruluğu %98,1'ken bütünleştirilen modelden elde edilen sonuçların performansı %97,86 olarak %0,24'lük bir düşüşle korunmuştur ve tespit (inference) süresi de 34,94ms olarak elde edilmiştir. Gerçek zamanlı bir görüntüde ise bu süre yaklaşık olarak saniyede 30 çerçeve olarak karşılık bulmaktadır. Elde edilen bu hız, oluşturulan mimarinin baskılı devre üretim bantlarında bulunan bir gömülü sisteme dahil edilebileceğini göstermektedir. Ayrıca tasarlanan mimaride kullanılan ağırlık değerleri harici bir hafıza biriminde tutulmaktadır. Bu durum, başka bir veriseti ile eğitilmiş olan herhangi bir YOLOv11n ağırlık grubunu ilgili hafıza birimine yazıldığı taktirde aynı modelin farklı ağırlık seti ile çalışabileceği ve başka nesne tespit hedefleri için kullanılabilir olduğu anlamına gelmektedir.

Kullanılan YOLOv11n modelinin daha üst modellerinin kullanılması, donanım mimarisinin optimize edilerek kaynak tüketiminin azaltılması, daha fazla kaynağa sahip bir FPGA modelinin seçilmesi veya daha hızlı haberleşmeye sahip (IFC – Integrated Flash Controller gibi) dışarıdan bağlantılı hafıza birimleri kullanılması gerçek zamanlı uygulamalarda hem performans hem de hız açısından önemli gelişmeler sunabilir. Oluşturulan mimaride kayan noktalı gösterim olarak 16 bit kullanılması performans açısından bir düşüşe sebep olmuştur. Aynı zamanda ağırlıkların depolandığı hafıza biriminin haberleşmesindeki yavaşlıktan dolayı FPGA içerisinde yüksek boyutlu FIFO kullanım ihtiyacı olmuştur. Kayan noktalı gösterimin 32 bit olarak kullanılması performansı yükseltecek fakat hız ve kaynak tüketimi açısından dezavantajlı olacaktır. Bu dezavantajların giderilmesi için ise ağırlıkların daha hızlı okunarak sisteme verilmesi etkili olacaktır. Ayrıca, model içerisinde az kullanılan ağırlıkların budama teknikleri ile kaldırılması işlem yoğunluğunu azaltarak hız kazandırabilir.

Gelecek çalışmalarda her derin öğrenme modelinde olduğu gibi veri setinin genişletilmesi ve çeşitlendirilmesi performans iyileştirmesi açısından faydalı olacaktır. Ek olarak tasarlanan modelin kaynak optimizasyonu ve yeniden şekillendirilerek fazlalık işlemlerin çıkartılması modelin çıkarım yapma süresini azaltacaktır. Ayrıca daha üst seviye YOLOv11 modelleri kullanılarak performans artımı sağlanabilir. Entegrasyon açısından yapılabilecek yenilik ise modele sonraki süreç (post-process) yapıları eklenerek sınıflandırma sonuçlarının elde edilmesi sağlanabilir. Son olarak elde edilen sonuçları çerçeve üzerine işleyen ve bu görüntüyü dışarıya aktaran birtakım modül eklenerek nesne tespit sonuçları direkt olarak FPGA'dan görüntü çıktısı olarak verilebilir.

Son olarak, bu tez çalışması YOLOv11 derin öğrenme modelinin FPGA'ya uygulanabilirliğini ve derin öğrenme tabanlı baskılı devre kusur tespitinin gömülü sistemlere entegrasyonunun gerçekleştirilmesiyle endüstride bir kullanım alanına sahip olabileceğini göstermektedir.

Gelecek çalışmaların hayata geçirilmesi ve önerilen mimarinin entegrasyonu ile üretilen baskılı devrelerdeki kusurların tespitine katkı sağlayarak üretim süreçlerinde önemli gelişmeler sağlayacaktır.

# KAYNAKLAR


[1] H. Suzuki, "Printed circuit board," U.S. Patent 4,640,866, Mar. 16, 1987.

[2] H. Matsubara, M. Itai, and K. Kimura, "Printed circuit board," U.S. Patent 6,573,458, Sept. 12, 2003.

[3] J. A. Magera and G. J. Dunn, *The Printed Circuit Designer's Guide to Flex and Rigid-Flex Fundamentals*, Motorola Solutions Inc., U.S. Patent 7,459,202, Aug. 21, 2008.

[4] H. S. Cho, J. G. Yoo, J. S. Kim, and S. H. Kim, "Printed circuit board," U.S. Patent 8,159,824, Samsung Electro Mechanics Co. Ltd., 2012.

[5] H. Yu, Q. Lin, and C. Liu, "Design and implementation of embedded PCB defect detection system based on FPGA," in *Proc. 2023 IEEE 7th Information Technology and Mechatronics Engineering Conf. (ITOEC)*, Chongqing, China, 2023, pp. 530–535, doi: 10.1109/ITOEC57671.2023.10292026.

[6] N. K. Khalid, Z. Ibrahim, *An Image Processing Approach towards Classification of Defects on Printed Circuit Board*, Ph.D. dissertation, Univ. Technology Malaysia, Johor, Malaysia, 2007.

[7] V. A. Adibhatla et al., "Printed circuit board defect detection," *Math. Biosci. Eng.*, vol. 18, no. 4, pp. 4411–4428, 2021. doi: 10.3934/mbe.2021223.

[8] P. S. Malge, "PCB defect detection, classification and localization using mathematical morphology and image processing tools," *Int. J. Comput. Appl.*, vol. 87, pp. 40–45, 2014. doi: 10.5120/15240-3782.

[9] Y. Takada, T. Shiina, H. Usami, and Y. Iwahori, "Defect detection and classification of electronic circuit boards using keypoint extraction and CNN features," in *Proc. 9th Int. Conf. Pervasive Patterns and Applications*, 2017, pp. 113–116.

[10] D. B. Anitha and R. Mahesh, "A survey on defect detection in bare PCB and assembled PCB using image processing techniques," in *Proc. 2017 Int. Conf. Wireless Commun., Signal Process. Netw. (WiSPNET)*, 2017, pp. 39–43.

doi: 10.1109/WiSPNET.2017.8299715.

[11] A. J. Crispin and V. Rankov, "Automated inspection of PCB components using a genetic algorithm template-matching approach," *Int. J. Adv. Manuf. Technol.*, vol. 35, pp. 293–300, 2007. doi: 10.1007/s00170-006-0730-0.

[12] F. Raihan and W. Ce, "PCB defect detection using OpenCV with image subtraction method," in *Proc. 2017 Int. Conf. Inf. Manage. Technol. (ICIMTech)*, 2017, pp. 204–209. doi: 10.1109/ICIMTech.2017.8273538.

[13] M. Aydın and F. Kaçar, "Detection of printed circuit board faults with FPGA-based real-time image processing," *Iran J. Comput. Sci.*, vol. 6, no. 4, pp. 419–430, Dec. 2023. doi: 10.1007/s42044-023-00149-6.

[14] M. H. Annaby, Y. M. Fouda ve M. A. Rushdi, "Improved normalized cross-correlation for defect detection in printed-circuit boards," *IEEE Trans. Semicond. Manuf.*, vol. 32, no. 2, pp. 199–211, 2019. doi:10.1109/TSM.2019.2911062.

[15] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," *arXiv preprint*, arXiv:1512.03385, 2015.

[16] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint*, arXiv:1409.1556, 2014.

[17] J. Redmon, S. Divvala, R. Girshick, and A. Farhadi, "You Only Look Once: Unified, real-time object detection," in *Proc. 2016 IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Las Vegas, NV, USA, 2016, pp. 779–788, doi: 10.1109/CVPR.2016.91.

[18] H. Yu, Q. Lin, and C. Liu, "Design and implementation of embedded PCB defect detection system based on FPGA," in *Proc. 2023 IEEE 7th Information Technology and Mechatronics Engineering Conf. (ITOEC)*, Chongqing, China, 2023, pp. 530–535, doi: 10.1109/ITOEC57671.2023.10292026.

[19] J. H. Park, Y. S. Kim, H. Seo, and Y. J. Cho, "Analysis of training deep learning models for PCB defect detection," *Sensors*, vol. 23, no. 5, 2023. doi: 10.3390/s23052766.

[20] J. Lim, J. Lim, V. M. Baskaran, and X. Wang, "A deep context learning based PCB defect detection model with anomalous trend alarming system," *Results Eng.*, vol. 17, p. 100968, 2023. doi: 10.1016/j.rineng.2023.100968

[21] Y. Pan, L. Zhang, and Y. Zhang, "Rapid detection of PCB defects based on YOLOx-Plus and FPGA," *IEEE Access*, vol. 12, pp. 61343–61358, 2024, doi: 10.1109/ACCESS.2024.3387947.

[22] R. Khanam and M. Hussain, "YOLOv11: An overview of the key architectural enhancements," *arXiv preprint*, arXiv:2024, 2024.

[23] A. R. Ünsal, "FPGA kullanarak şablon eşleme temelli yüz tanıyan donanımın gerçekleştirilmesi," M.S. thesis, Sakarya Uygulamalı Bilimler Üniversitesi, Sakarya, Turkey, 2023.

[24] E. Yuk, S. Park, C.-S. Park, and J.-G. Baek, "Feature-learning-based printed circuit board inspection via speeded-up robust features and random forest," *Appl. Sci.*, vol. 8, p. 932, 2018.

[25] J. Kim, J. Ko, H. Choi, and H. Kim, "Printed circuit board defect detection using deep learning via a skip-connected convolutional autoencoder," *Sensors*, vol. 21, no. 15, 2021. doi: 10.3390/s21154968.

[26] W. Huang ve P. Wei, "A PCB dataset for defects detection and classification," arXiv preprint, arXiv:1901.08204, 2019. doi:10.48550/arXiv.1901.08204

[27] A. Bochkovskiy, C. Y. Wang, ve H. Y. M. Liao, "YOLOv4: Optimal speed and accuracy of object detection," arXiv preprint, arXiv:2004.10934, 2020. doi:10.48550/arXiv.2004.10934

[28] X. Wu, Y. Ge, Q. Zhang, and D. Zhang, "PCB defect detection using deep learning methods," in *Proc. 2021 IEEE 24th Int. Conf. Comput. Supported Cooperative Work in Design (CSCWD)*, Dalian, China, 2021, pp. 873–876, doi: 10.1109/CSCWD49262.2021.9437846.

[29] T. Y. Lin et al., "Feature pyramid networks for object detection," in Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR), Honolulu, HI, USA, Jul. 2017, pp. 936–944.

doi:10.1109/CVPR.2017.106

[30] S. Ren, K. He, R. Girshick et al., “Faster R-CNN: Towards real-time object detection with region proposal networks,” IEEE Trans. Pattern Anal. Mach. Intell., vol. 39, no. 6, pp. 1137–1149, 2017. doi:10.1109/TPAMI.2016.2577031

[31] S. Tang, F. He, X. Huang, and J. Yang, "Online PCB defect detector on a new PCB defect dataset," *arXiv preprint*, arXiv:2019. doi:10.48550/arXiv.1902.06197.

[32] G. Jocher et al., "ultralytics/yolov5: v6.0 - YOLOv5n ’Nano’ models, Roboflow integration, TensorFlow export, OpenCV DNN support," *Zenodo*, doi: 10.5281/zenodo.5563715, 2021.

[33] M. Shen et al., "Defect detection of printed circuit board assembly based on YOLOv5," *Sci. Rep.*, vol. 14, no. 1, p. 19287, Aug. 2024. doi: 10.1038/s41598-024-70176-1.

[34] P. A. Salunke, S. N. Sherkar, and C. S. Arya, "PCB (Printed Circuit Board) fault detection using machine learning," *Int. J. Comput. Sci. Mobile Comput.*, vol. 10, no. 2, pp. 54–56, 2021.

[35] V. A. Adibhatla et al., "Printed circuit board defect detection," *Math. Biosci. Eng.*, vol. 18, no. 4, pp. 4411–4428, 2021, doi: 10.3934/mbe.2021223.

[36] R. Ding, L. Dai, G. Li, and H. Liu, "TDD-net: A tiny defect detection network for printed circuit boards," *CAAI Trans. Intell. Technol.*, vol. 4, no. 2, pp. 110–116, 2019, doi: 10.1049/trit.2019.0019.

[37] X. Liao et al., "YOLOv4-MN3 for PCB surface defect detection," *Appl. Sci.*, vol. 11, p. 11701, 2021, doi: 10.3390/app11241170.

[38] A. Bhattacharya and S. G. Cloutier, "End-to-end deep learning framework for printed circuit board manufacturing defect classification," *Sci. Rep.*, vol. 12, p. 12559, 2022, doi: 10.1038/s41598-022-16302-3.

[39] W. Huang, P. Wei, and M. Zhang, "HRIPCB: A challenging dataset for PCB defects detection and classification," *J. Eng.*, 2020, doi: 10.1049/joe.2019.1183.

[40] V. Kaya and İ. Akgül, "Detection of defects in printed circuit boards with machine learning and deep learning algorithms," *Eur. J. Sci. Technol.*, no. 41, pp. 183–186, 2022. doi: /10.31590/ejosat.1178188.

[41] N. Dalal and B. Triggs, "Histograms of oriented gradients for human detection," in *Proc. IEEE Comput. Vis. Pattern Recognit. (CVPR'05)*, vol. 1, pp. 886–893, 2005, doi: 10.1109/CVPR.2005.177.

[42] T. Cover and P. Hart, "Nearest neighbor pattern classification," *IEEE Trans. Inf. Theory*, vol. 13, no. 1, pp. 21–27, 1967, doi: 10.1109/TIT.1967.1053964.

[43] B. E. Boser, I. M. Guyon, and V. N. Vapnik, "A training algorithm for optimal margin classifiers," in *Proc. 5th Annu. Workshop Comput. Learn. Theory*, pp. 144–152, 1992, doi: 10.1145/130385.130401.

[44] J. Lim, J. Lim, V. M. Baskaran, and X. Wang, "A deep context learning based PCB defect detection model with anomalous trend alarming system," *Results Eng.*, vol. 17, p. 100968, 2023, doi: 10.1016/j.rineng.2023.100968.

[45] J. Redmon and A. Farhadi, "YOLOv3: An incremental improvement," *arXiv preprint*, arXiv:1804.02767, 2018. [Online]. Available: https://arxiv.org/abs/1804.02767.

[46] Y. Pan, L. Zhang, and Y. Zhang, "Rapid detection of PCB defects based on YOLOx-Plus and FPGA," *IEEE Access*, 2024, doi: 10.1109/ACCESS.2024.3387947.

[47] İ. Budak, *FPGA Tabanlı PCB Hata Tespitinde Derin Öğrenme Uygulaması*, M.S. thesis, Marmara Üniversitesi, Fen Bilimleri Enstitüsü, Elektrik-Elektronik Müh. A.B.D., İstanbul, Turkey, 2024.

[48] C. M. Bishop, *Pattern Recognition and Machine Learning*, Springer, 2006.

[49] E. Alpaydin, *Introduction to Machine Learning*, MIT Press, 2020.

[50] C. Janiesch, P. Zschech, and K. Heinrich, "Machine learning and deep learning," *Electron. Markets*, vol. 31, pp. 685–695, 2021, doi: 10.1007/s12525-021-00475-2.

[51] H. J. Weerts, A. C. Mueller, and J. Vanschoren, "Importance of tuning

hyperparameters of machine learning algorithms," *arXiv preprint*, arXiv:2007.07588, Jul. 2020, doi: 10.48550/arXiv.2007.07588.

[52] P. Probst, A. L. Boulesteix, and B. Bischl, "Tunability: Importance of hyperparameters of machine learning algorithms," *J. Mach. Learn. Res.*, vol. 20, no. 53, pp. 1–32, 2019, doi: 10.48550/arXiv.1802.09596.

[53] M. Emeç and M. Ozcanhan, "Makine öğrenmesi algoritmalarında hiper parametre belirleme," *Makine Öğrenmesi ve Optimizasyon Çalışmaları Dergisi*, vol. 39, 2023, doi: 10.59287/mocc.39.

[54] N. Lavesson and P. Davidsson, "Quantifying the impact of learning algorithm parameter tuning," in *Proc. 21st Nat. Conf. Artif. Intell. (AAAI'06)*, vol. 1, AAAI Press, pp. 395–400, 2006. [Online]. Available: http://dl.acm.org/citation.cfm?id=1597538.1597602.

[55] N. Gupta et al., "Data quality for machine learning tasks," in *Proc. 27th ACM SIGKDD Conf. Knowledge Discovery Data Mining (KDD '21)*, Aug. 2021, pp. 4040–4041, doi: 10.1145/3447548.3470817.

[56] E. Balkan, *Traffic Signs Recognition System for Autonomous Vehicles*, M.S. thesis, Dept. Electr. Electron. Eng., Istanbul Arel Univ., Istanbul, Turkey, 2022.

[57] I. Muhammad and Z. Yan, "Supervised machine learning approaches: A survey," *ICTACT J. Soft Comput.*, vol. 5, no. 3, pp. 946–952, 2015, doi: 10.21917/ijsc.2015.0133.

[58] Roboflow, "PCBYOLO8 Dataset," *Roboflow Universe*, Jan. 2024.

[59] Roboflow, "Obj detection-pcb-defects-yolov8 Dataset," *Roboflow Universe*, Jan. 2024.

[60] M. Ş. Akçay, İ. Koyuncu, M. Alçin, and M. Tuna, "FPGA tabanlı LogSig ve TanSig transfer fonksiyonlarının IQ-Math sayı standardında tasarımı ve gerçekleştirilmesi," *J. Mater. Mechat. A*, vol. 3, no. 2, pp. 225–239, 2022, doi: 10.55546/jmm.1094815.

[61] S. Hasçelik, *Konvolüsyonel sinir ağı kullanılarak trafik işaretlerini gerçek zamanlı*

*bulma ve tanıma*, M.S. thesis, Dept. Electr. Electron. Eng., Trakya Univ., Edirne, Turkey, 2021.

[62] A. M. Turing, "Computing machinery and intelligence," *Mind*, vol. 59, pp. 433–460, 1950, doi: 10.1093/mind/LIX.236.433.

[63] D. C. Plaut and G. E. Hinton, "Learning sets of filters using back propagation," *Comput. Speech Lang.*, vol. 2, pp. 35–61, 1987.

[64] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proc. IEEE*, vol. 86, no. 11, pp. 2278–2324, Nov. 1998, doi: 10.1109/5.726791.

[65] S. Albawi, T. A. Mohammed, and S. Al-Zawi, "Understanding of a convolutional neural network," in *Proc. 2017 Int. Conf. Eng. Technol. (ICET)*, Antalya, Turkey, Aug. 2017, pp. 1–6.

[66] I. Goodfellow, Y. Bengio, and A. Courville, *Deep Learning*, MIT Press, 2016.

[67] G. Karataş, O. Demir, and O. K. Sahingoz, "Deep learning in intrusion detection systems," in *Proc. Int. Congr. Big Data Deep Learning Fighting Cyber*, Dec. 2018.

[68] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, ve R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," J. Mach. Learn. Res., vol. 15, pp. 1929–1958, 2014. doi:10.5555/2627435.2670313 .

[69] B. Rohrer, "How convolutional neural networks work," [Online]. Available: https://e2eml.school/how_convolutional_neural_networks_work.html, Mar. 8, 2025. [Accessed: Apr. 2025].

[70] V. Tümen, Ö. Yıldırım, and B. Ergen, "A convolutional neural network model for road flow direction detection," *J. Intell. Syst. Appl.*, vol. 2, no. 2, pp. 94–99, 2019, doi: 10.54856/jiswa.201912072.

[71] T. Serre, L. Wolf, S. Bileschi, M. Riesenhuber, and T. Poggio, "Robust object recognition with cortex-like mechanisms," *IEEE Trans. Pattern Anal. Mach. Intell.*, no. 3, pp. 411–426, 2007.

[72] Ö. İnik and E. Ülker, "Derin öğrenme ve görüntü analizinde kullanılan derin öğrenme modelleri," *GBAD*, vol. 6, no. 3, pp. 85–104, Dec. 2017.

[73] M. Turan and E. Arığ, "Video duygu analizi," *Avrupa Bilim ve Teknoloji Dergisi*, no. EJOSAT Ek Özel Sayı (HORA), pp. 34–41, 2021.

[74] V. Nair and G. Hinton, "Rectified linear units improve restricted Boltzmann machines," in *Proc. 27th Int. Conf. Mach. Learn.*, Jun. 2010, pp. 807–814.

[75] Y. Gal and Z. Ghahramani, "Dropout as a Bayesian approximation: Representing model uncertainty in deep learning," in *Proc. 33rd Int. Conf. Mach. Learn.*, Jun. 2015.

[76] S. Elfwing, E. Uchibe, and K. Doya, "Sigmoid-weighted linear units for neural network function approximation in reinforcement learning," *Neural Netw.*, vol. 107, pp. 3–11, 2018, doi: 10.1016/j.neunet.2017.12.012.

[77] M. D. Zeiler and R. Fergus, "Visualizing and understanding convolutional networks," *CoRR*, vol. abs/1311.2901, 2013. [Online]. Available: http://arxiv.org/abs/1311.2901, doi: 10.1007/978-3-319-10590-1_53.

[78] J. Weston, F. Ratle, H. Mobahi, and R. Collobert, "Deep learning via semi-supervised embedding," in *Neural Networks: Tricks of the Trade*, 2nd ed., G. Montavon, G. B. Orr, and K.-R. Müller, Eds. Berlin, Heidelberg: Springer, 2012, pp. 639–655, doi: 10.1007/978-3-642-35289-8_34.

[79] M. S., *İnsansız hava aracı görüntülerinden derin öğrenme yöntemleriyle nesne tanıma*, M.S. thesis, Maltepe Univ., Lisansüstü Eğitim Enstitüsü, İstanbul, 2021.

[80] O. M. Abdulhadi, *An Efficient FPGA Implementation of CNN Specialized in Image Recognition for Breast Cancer*, M.S. thesis, İstanbul Gelişim Univ., İstanbul, 2022.

[81] N. Srivastava, G. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: A simple way to prevent neural networks from overfitting," *J. Mach. Learn. Res.*, vol. 15, pp. 1929–1958, Jun. 2014.

[82] İ. Öztel, *Kısmi ve tam yüz görüntüleri üzerinde makine öğrenmesi yöntemleriyle yüz*

*ifadesi tespiti*, Ph.D. dissertation, Sakarya Univ., Fen Bilimleri Enstitüsü, Sakarya, 2018.

[83] F. Doğan and İ. Türkoğlu, "Derin öğrenme algoritmalarının yaprak sınıflandırma başarımlarının karşılaştırılması," *SAUCIS*, vol. 1, no. 1, pp. 10–21, 2018.

[84] Y. Gao, Y. Xin, H. Yang, and Y. Wang, "A lightweight anti-unmanned aerial vehicle detection method based on improved YOLOv11," *Drones*, vol. 9, no. 12, p. 11, Dec. 2024, doi: 10.3390/drones9010011.

[85] F. Kateb, M. M. Monowar, M. A. Hamid, A. Ohi, and M. Ph. D., "FruitDet: Attentive feature aggregation for real-time fruit detection in orchards," *Agronomy*, vol. 11, no. 12, p. 2440, Nov. 2021, doi: 10.3390/agronomy11122440.

[86] T. Y. Lin et al., "Feature pyramid networks for object detection," in *Proc. IEEE Conf. Comput. Vis. Pattern Recognit. (CVPR)*, Honolulu, HI, USA, 2017, pp. 936–944, doi: 10.1109/CVPR.2017.106.

[87] N. Rao, "YOLOv11 architecture explained: Next-level object detection with enhanced speed and accuracy," *Medium*, Oct. 22, 2024. [Online]. Available: https://medium.com/@ nikhil- rao-20/yolov11-explained-next-level-object-detection-with-enhanced-speed-and-accuracy-2dbe2d376f71. [Accessed: Apr. 3, 2025].

[88] L. He, Y. Zhou, L. Liu, and J. Ma, "Research and application of YOLOv11-based object segmentation in intelligent recognition at construction sites," *Buildings*, vol. 14, p. 3777, Nov. 2024, doi: 10.3390/buildings14123777.

[89] N. Jegham, C. Y. Koh, M. Abdelatti, and A. Hendawi, "YOLO evolution: A comprehensive benchmark and architectural review of YOLOv12, YOLO11, and their previous versions," *arXiv preprint*, arXiv:2411.00201, 2025. [Online]. Available: https://arxiv.org/abs/2411.00201.

[90] S. Ruuska et al., "Evaluation of the confusion matrix method in the validation of an automated system for measuring feeding behavior of cattle," *Behav. Process.*, vol. 148, pp. 56–62, 2018, doi: 10.1016/j.beproc.2018.01.004.

[91] R. Padilla, S. L. Netto, and E. A. B. da Silva, "A survey on performance metrics for

object-detection algorithms," in *Proc. 2020 Int. Conf. Syst., Signals Image Process. (IWSSIP)*, Niteroi, Brazil, 2020, pp. 237–242, doi: 10.1109/IWSSIP48289.2020.9145130.

[92] P. Jaccard, "Etude comparative de la distribution florale dans une portion des Alpes et des Jura," *Bull. Soc. Vaudoise Sci. Nat.*, vol. 37, pp. 547–579, 1901.

[93] E. Tlelo-Cuautle, J. Rangel-Magdaleno, and L. de la Fraga, "Introduction to field-programmable gate arrays," in *Field-Programmable Gate Arrays*, Springer, 2016, pp. 1–32, doi: 10.1007/978-3-319-34115-6_1.

[94] F. Karataş, *VHDL ile FPGA-Tabanlı EKG Simülatörü Tasarımı*, M.S. thesis, Afyon Kocatepe Univ., Fen Bilimleri Enstitüsü, Afyonkarahisar, Turkey, 2021.

[95] İ. Erbaş, *FPGA implementation of machine learning algorithms for vibrotactile feedback in prostheses*, M.S. thesis, Biyomedikal Müh. Enstitüsü, Boğaziçi Univ., İstanbul, 2021.

[96] S. Dereli, *Yapısal devre tasarımı, FPGA ile gömülü sistemler ve sayısal devre tasarımı*, Nobel, 2019, pp. 155–171.

[97] N. Shibata, M. Watanabe, and Y. Tanabe, "A current-sensed high-speed and low-power first-in-first-out memory using a wordline/bitline-swapped dual-port SRAM cell," *IEEE J. Solid-State Circuits*, vol. 37, no. 6, pp. 735–750, Jun. 2002, doi: 10.1109/JSSC.2002.1004578.

[98] Y. Zhang, C. Yi, J. Wang, and J. Zhang, "Asynchronous FIFO implementation using FPGA," in *Proc. 2011 Int. Conf. Electron. Optoelectron.*, Dalian, 2011, pp. V3-207–V3-209, doi: 10.1109/ICEOE.2011.6013339.

[99] "IEEE standard for floating-point arithmetic," *IEEE Std 754-2019 (Revision of IEEE 754-2008)*, pp. 1–84, Jul. 22, 2019, doi: 10.1109/IEEESTD.2019.8766229.

[100] AMD, *Floating-Point Operator v7.1 LogiCORE IP Product Guide*, ver. 7.1, Dec. 20, 2023. [Online]. Available: https://docs.amd.com/v/u/en-US/pg060-floating-point. [Accessed: Apr. 15, 2025].

[101] AMD, *AXI Reference Guide*, UG1037, ver. 4.0, Jul. 15, 2017. [Online]. Available: https://docs.amd.com/v/u/en-US/ug1037-vivado-axi-reference-guide. [Accessed: Apr. 19, 2025].

[102] VESA, *Industry Standards and Guidelines for Computer Display Monitor Timing (DMT)*, Video Electron. Standards Assoc., Newark, CA, USA, Feb. 2013. [Online]. Available: https://glenwing.github.io/docs/VESA-DMT-1.13.pdf.