

T.C.
BALIKESİR ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI



NİCELENMİŞ DERİN ÖĞRENME AĞLARI İÇİN FPGA TABANLI
HIZLANDIRICI TASARIMI

MUSTAFA TAŞCI
DOKTORA TEZİ

JÜRİ ÜYELERİ: Prof.Dr. Ayhan İSTANBULLU (Tez Danışmanı)

Prof.Dr. Metin DEMİRTAŞ

Doç.Dr. Metin BİLGİN

Doç.Dr. İbrahim ARPACI

Dr.Öğr. Üyesi Erdem İLTEN

BALIKESİR, ARALIK - 2022

ETİK BEYAN

Balıkesir Üniversitesi Fen Bilimleri Enstitüsü Tez Yazım Kurallarına uygun olarak tarafımda hazırlanan “**Nicelenmiş Derin Öğrenme Ağları İçin FPGA Tabanlı Hızlandırıcı Tasarımı**” başlıklı tezde;

- Tüm bilgi ve belgeleri akademik kurallar çerçevesinde elde ettiğimi,
- Kullanılan veriler ve sonuçlarda herhangi bir değişiklik yapmadığımı,
- Tüm bilgi ve sonuçları bilimsel araştırma ve etik ilkelere uygun şekilde sunduğumu,
- Yararlandığım eserlere atıfta bulunarak kaynak gösterdiğimi,

beyan eder, aksinin ortaya çıkması durumunda her türlü yasal sonucu kabul ederim.

Mustafa TAŞCI

Bu tez çalışması Balıkesir Üniversitesi tarafından (BAP 2020/091) numaralı proje ile desteklenmiştir.

ÖZET

**NİCELENMİŞ DERİN ÖĞRENME AĞLARI İÇİN FPGA TABANLI
HIZLANDIRICI TASARIMI
DOKTORA TEZİ
MUSTAFA TAŞCI
BALIKESİR ÜNİVERSİTESİ FEN BİLİMLERİ ENSTİTÜSÜ
ELEKTRİK-ELEKTRONİK MÜHENDİSLİĞİ ANABİLİM DALI
(TEZ DANIŞMANI: PROF.DR. AYHAN İSTANBULLU)
BALIKESİR, ARALIK - 2022**

Hayatı kolaylaştıran akıllı mobil cihazların geliştirilmesi günümüz teknolojisinin en önemli uygulama alanlarından biridir. Bu tür teknolojiler geliştirilirken derin öğrenme tekniği oldukça yaygın olarak kullanılmaktadır. Derin öğrenme algoritmaları milyarlarca kayar noktalı (GFLOPS) matematiksel işlem ve yüksek hafıza bant genişliğine ihtiyaç duymaktadır. Sıralı işlem yapan klasik işlemciler derin öğrenme uygulamalarında yetersiz kaldıkları için grafik işleme birimleri (GPU), özel entegre devreler (ASIC) ve alan etkili kapı dizileri (FPGA) gibi donanım hızlandırıcılar derin öğrenme uygulamalarında sıklıkla kullanılmaktadır. Özellikle taşınabilir cihazlarda derin öğrenme uygulamalarının gerçekleştirilmesinde FPGA hızlandırıcılar ön plana çıkmaktadır. Bu çalışmada MLP ve Lenet modelleri için Xilinx PYNQ Z1 geliştirme kartı üzerinde hızlandırıcı donanımlar geliştirilmiştir. Bu tez çalışmasında, farklı niceme seviyelerinde QNN hızlandırıcı geliştirmek için FINN çerçevesi kullanılmıştır. Farklı niceme ve katlama kombinasyonları kullanılarak MLP ve Lenet modelleri için 23 hızlandırıcı donanım oluşturulmuştur. Geliştirilen donanımlar, kaynak kullanımı, enerji tüketimleri, alan kullanımları ve başarı oranları açısından kıyaslanmıştır. Lenet modeli için W1A2 niceme, HM katlama seviyesinde geliştirilen hızlandırıcı FashionMNIST veri setini 6800 FPS hızında sınıflandırmıştır. MLP modeli için W1A1 niceme seviyesinde geliştirilen donanım MNIST veri setini 950 kFPS hızında sınıflandırmayı başarmıştır. Bu model için saniyede gerçekleştirilen işlem sayısı 1018 GOPs iken tüketilen enerji 2.05 W olarak ölçülmüştür. Güç verimliliği (watt başına işlem sayısı) yaklaşık 496 GOPs/W olarak hesaplanmıştır. Benzer çalışmalarla kıyaslandığında daha iyi performans elde edilmiştir.

ANAHTAR KELİMELER: Derin öğrenme, Hızlandırıcılar, FPGA, FINN, Brevitas

Bilim Kod / Kodları : 90520, 90542

Sayfa Sayısı : 84

ABSTRACT

FPGA-BASED ACCELERATOR DESIGN FOR QUANTIZED DEEP LEARNING NETWORKS

PH.D THESIS

MUSTAFA TAŞCI

BALIKESİR UNIVERSITY INSTITUTE OF SCIENCE

ELECTRICAL AND ELECTRONICS ENGINEERING

(SUPERVISOR: PROF.DR. AYHAN İSTANBULLU)

BALIKESİR, DECEMBER - 2022

The development of smart mobile devices that make life easier is one of the most important application areas of today's technology. While developing such technologies, deep learning (DL) technique is widely used. DL algorithms require billions of floating-point mathematical operations and high memory bandwidth. Hardware accelerators such as graphics processing units (GPU), special integrated circuits (ASIC) and field effect gate arrays (FPGA) are frequently used in DL applications since classical processors with sequential processing are insufficient in DL applications. FPGA accelerators hold a significant part in the realization of DL applications on portable devices. In this thesis, accelerator hardware was developed on the Xilinx PYNQ Z1 development board for multi-layer perception (MLP) and Lenet models. In this study, FINN framework is used to develop QNN accelerator at different quantization levels. Using different combinations of quantization and folding, 23 accelerators were created for the MLP and Lenet models. The accelerators have been compared in terms of resource usage, energy consumption, area usage and success rates. For the Lenet model, a precision of W1A2 classified the accelerator FashionMNIST dataset developed at the CNV high and FC medium folding levels at a speed of 6800 FPS. The hardware developed at the W1A1 quantization level for the MLP model has succeeded in classifying the MNIST dataset at 950 kFPS. The number of operations per second for this model is 1018 GOPs, while the energy consumed is 2,05W. Power efficiency is calculated at approximately 496 GOPs/W, in comparison to existing studies achieving a better performance.

KEYWORDS: Deep Learning, Accelerators, FPGA, FINN, Brevitas

Science Code / Codes: 90520, 90542

Page Number: 84

İÇİNDEKİLER

Sayfa

ÖZET	i
ABSTRACT	ii
İÇİNDEKİLER	iii
ŞEKİL LİSTESİ	v
TABLO LİSTESİ	vii
SEMBOL LİSTESİ	viii
KISALTMALAR LİSTESİ	ix
ÖNSÖZ	xi
1. GİRİŞ	1
2. FPGA TABANLI DERİN ÖĞRENME HIZLANDIRICILARI	4
3. ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ (FPGA)	10
3.1 PYNQ Yazılımı.....	10
3.2 Xilinx PYNQ Z1 FPGA Platformu.....	11
4. DERİN ÖĞRENME	12
4.1 Derin Öğrenme Çerçeveleri	12
4.2 Derin Öğrenme Veri Setleri	12
4.2.1 Mnist Veri Seti	12
4.2.2 FashionMNIST Veri Seti	13
4.3 Derin Öğrenme Modelleri.....	14
4.3.1 Çok Katmalı Sinir Ağları (MLP)	14
4.3.2 Evrişimsel Sinir Ağı (CNN).....	15
4.3.2.1 Evrişim Katmanı.....	16
4.3.2.2 Aktivasyon Fonksiyonu.....	17
4.3.2.3 Havuzlama Katmanı	17
4.3.2.4 Tam Bağlantı Katmanı	18
4.3.3 Nicelleştirilmiş Sinir Ağı (QNN).....	18
4.3.3.1 Xilinx Brevitas Kütüphanesi ile Nicelleştirme.....	20
4.3.4 İkileştirilmiş Sinir Ağı (BNN)	21
4.3.4.1 İkileştirme Fonksiyonları	22
4.3.4.2 BNN XNOR İşlevselliği.....	22
4.3.4.3 Gradyanların Yok Olması Problemi ve STE.....	24
4.4 Xilinx FINN Çerçevesi	25
4.4.1 FINN Katlama İşlemi.....	27
5. FPGA TABANLI DERİN ÖĞRENME HIZLANDIRICISI	29
5.1 Uygulama 1: HLS Tabanlı 8 bit Sabit Noktalı Hızlandırıcı	30
5.1.1 MLP Mimarisi.....	30
5.1.2 LENET-5 Mimarisi	31
5.1.3 FPGA IP Blokları ve Katmanlar	31
5.1.4 Uygulama1-MLP Modeli FPGA Bloğu	32
5.1.5 Uygulama1 - Lenet-5 FPGA Bloğu	33
5.1.6 MLP ve LENET IP Blokları ZYNQ Bağlantısı	33
5.1.7 Uygulama-1 Bulguları.....	34
5.2 Uygulama 2: FINN Tabanlı Hızlandırıcı	35

5.2.1 MLP Hızlandırıcı	36
5.2.1.1 MLP Model Oluşturma ve Eğitim	36
5.2.1.2 MLP Modeli Hızlandırıcı FINN Dönüşümleri	38
5.2.1.3 MLP Hızlandırıcı Donanımı Oluşturma.....	44
5.2.1.4 MLP Hızlandırıcı PYNQ Üzerinde Uygulama	46
5.2.1.5 MLP Hızlandırıcı Bulgular.....	47
5.2.2 Lenet Hızlandırıcı.....	52
5.2.2.1 Lenet Modeli Brevitas ile Oluşturma	52
5.2.2.2 Lenet Modeli FINN Dönüşümleri	53
5.2.2.3 FINN Lenet Hızlandırıcı Donanımı Oluşturma.....	56
5.2.2.4 Lenet Hızlandırıcı PYNQ Üzerinde Uygulama.....	58
5.2.2.5 Lenet Hızlandırıcı Uygulaması Bulguları	58
6. SONUÇLAR ve ÖNERİLER	69
7. KAYNAKLAR	73
EKLER	80
EK A: Lenet.py.....	80
EK B: MLP.py.....	81
EK C: trainer.py.....	83
ÖZGEÇMİŞ	84

ŞEKİL LİSTESİ

Sayfa

Şekil 1.1: ImageNet yarışması sonuçları.....	2
Şekil 1.2: ImageNet 2022 yılına kadar yapılan derin öğrenme projeleri TOP5 doğruluk oranları.....	2
Şekil 3.1: FPGA blok diyagramı [62].	10
Şekil 3.2: Xilinx PYNQ Z1 FPGA kartı.	11
Şekil 3.3: PYNQ Z1 Blok diyagramı.	11
Şekil 4.1: MNIST Veri seti.	13
Şekil 4.2: FashionMNIST Veri seti.....	14
Şekil 4.3: Sinir ağı hücresi.	14
Şekil 4.4: MLP Sinir ağı.....	15
Şekil 4.5: Lenet-5 Derin öğrenme modeli.....	16
Şekil 4.6: Girdi matrisi ile çekirdek matrisinin evrişim işlemi.	16
Şekil 4.7: (a) sigmoid, (b) Tanh, (c) ReLu ve (d) LeakyReLu aktivasyon fonksiyonları..	17
Şekil 4.8: Maksimum havuzlama katmanı.	17
Şekil 4.9: Tam bağlantılı katman.	18
Şekil 4.10: Niceleme işleminin derin öğrenme üzerindeki avantajları [48].....	19
Şekil 4.11: (a) PTQ ve (b) QAT niceleme akış şeması [63].	19
Şekil 4.12: (a) Standart evrişim (b) XNOR evrişim işlemi.	23
Şekil 4.13: XNOR-Net çalışması kazanç tablosu [24]......	24
Şekil 4.14: Derin öğrenme ileri ve geri yayılma blok diyagramı.....	24
Şekil 4.15: FINN donanım oluşturma akış şeması.....	25
Şekil 4.16: FINN dönüşüm fonksiyon işlemleri.	26
Şekil 4.17: FINN çerçevesi SWU ve MVTU birimleri akış şeması [52]......	27
Şekil 4.18: PE/SIMD ile Matris paralelleştirme.	27
Şekil 5.1: Çok katmanlı sinir ağı (MLP) modeli.....	30
Şekil 5.2: Uygulamada kullanılan Lenet-5 derin öğrenme modeli.	31
Şekil 5.3: Evrişim katmanı IP bloğu.	31
Şekil 5.4: Havuzlama katmanı IP bloğu.....	32
Şekil 5.5: Tam bağlantılı katman IP bloğu.....	32
Şekil 5.6: MLP modeli IP bloğu.	32
Şekil 5.7: Lenet-5 modeli IP bloğu.	33
Şekil 5.8: HLS standart hızlandırıcı donanımı.....	33
Şekil 5.9: 8 bit MLP ve Lenet modelleri için FPGA ve CPU süre doğruluk grafiği.	34
Şekil 5.10: MLP ve Lenet modeline göre FPGA/CPU hızlandırma oranları grafiği.....	35
Şekil 5.11: MLP hızlandırıcı blok şeması.	36
Şekil 5.12: MLP modeli MNIST ve FashionMNIST eğitim aşaması.....	38
Şekil 5.13: Brevitas ile oluşturulmuş MLP modeli Netron görünümü.	39
Şekil 5.14: FINN TidyUp ve PrePost dönüşümleri sonucu oluşan MLP modeli.....	41
Şekil 5.15: FINN Streamline dönüşümleri sonucu oluşan MLP modeli.....	41
Şekil 5.16: HLS hazırlık dönüşümleri sonucu oluşan modeller.....	42
Şekil 5.17: (a) HLS MVU dönüşümü, (b) DataFlow dönüşü sonucu oluşan modeller.	43
Şekil 5.19: PE/SIMD katlama parametreleri eklenmiş model.	44
Şekil 5.20: (a) MLP veri akışı modeli (b) MLP hızlandırıcı IP bloğu.	45
Şekil 5.21: MLP hızlandırıcı ve ZYNQ işletim sistemi bağlantısı.	45
Şekil 5.22: Verimlilik testi çıktısı.	46

Şekil 5.23: MLP hızlandırıcı niceleme ve katlama seviyelerine göre kaynak kullanımı ve doğruluk grafiği.	48
Şekil 5.24: Fiziksel ölçümlerde kullanılan USB güç ölçer.	49
Şekil 5.25: Güç tüketimi ölçüm ve Vivado tahmin sonuçları grafiği.	49
Şekil 5.26: (a) MLP W1A1 hızlandırıcı (b)MLP W1A2 hızlandırıcı (c) MLP W2A2 hızlandırıcı güç tüketimi adgılımlı Vivado tahminleri.	50
Şekil 5.27: (a) MLP 1W1A (b) MLP 1W2A (c) MLP2W2A Hızlandırıcıları FPGA alan kullanımları.	51
Şekil 5.28: Lenet-5 modeli blok şeması.	53
Şekil 5.29: Lenet modeli üç farklı hızlandırıcının eğitim aşaması doğruluk ve kayıp grafiği.	53
Şekil 5.30: Brevitas ile eğitilmiş Lenet modeli onnx görüntüsü.	54
Şekil 5.31: Evrişim, havuzlama ve tam bağlantı katmanlarında veri akışı [52].	55
Şekil 5.32: Evrişim işleminin Im2Col işlemine dönüştürülmesi.	55
Şekil 5.33: StreamLine ve katlama parametrelerinin eklendiği dönüşüm sonrası modelin görünümü.	56
Şekil 5.34: (a) Lenet veri akışı modeli (b) Lenet hızlandırıcı IP bloğu.	57
Şekil 5.35: Lenet hızlandırıcı ve ZYNQ işletim sistemi bağlantısı.	57
Şekil 5.36: Lenet hızlandırıcı niceleme ve katlama seviyelerine göre kaynak kullanım grafiği.	59
Şekil 5.37: Niceleme seviyelerine göre Lenet modeli hızlandırıcıların güç tüketimi ölçüm ve tahmin sonuçları grafiği.	60
Şekil 5.38: Lenet modeli hızlandırıcılar (a)W1A1, (b) W1A2 ve (c) W2A2 niceleme seviyelerine göre güç analizi.	61
Şekil 5.39: Lenet hızlandırıcı donanımları (a) W1A1, (b) W1A2 ve (c) W2A2 alan kullanımları.	62
Şekil 5.40: Evrişim ve Tam bağlantılı katman katlama faktörlerinin hızlandırıcılar üstünde kaynak kullanımı ve verime etki grafiği (Performans/Kaynak kullanımı açısından en iyi konfigürasyon 8 numaralı konfigürasyondur).	65
Şekil 5.41: Mantıksal ve bellek kaynaklarının katlama seviyelerine göre performans grafikleri.	67

TABLO LİSTESİ

Sayfa

Tablo 4.1: Derin öğrenme çerçeveleri karşılaştırma tablosu.....	12
Tablo 4.2: Pytorch fonksiyonlarının Brevitas karşılıkları.....	21
Tablo 5.1: Geliştirilen hızlandırıcı donanımlar özet tablosu.....	29
Tablo 5.2: HLS tabanlı 8 bit sabit noktalı FPGA hızlandırıcı test planı.....	30
Tablo 5.3: Üç farklı niceleme seviyesinde MLP ve Lenet modeli için oluşturulan hızlandırıcı test planı.....	36
Tablo 5.4: Niceleme türü belirleme algoritması Pseudo kodu.....	37
Tablo 5.5: Pytorch / Brevitas MLP bloğu oluşturma kodu.....	37
Tablo 5.6: Brevitastan FINN'a ONNX Dönüşüm Kodu.....	39
Tablo 5.7: Model'e katlama parametreleri ekleme kodu.....	43
Tablo 5.8: Zynq hızlandırıcı ve Pynq sürücü oluşturma kodu.....	44
Tablo 5.9: MLP hızlandırıcı donanımı test planı.....	46
Tablo 5.10: MLP model katlama parametreleri.....	47
Tablo 5.11: MLP hızlandırıcı niceleme / katlama oranlarına göre kaynak kullanımı verim tablosu.....	47
Tablo 5.12: MLP hızlandırıcıların Vivado programı güç tüketim tahminleri tablosu.....	50
Tablo 5.13: Lenet modeli pytorch / brevitastan kodu.....	52
Tablo 5.14: Lenet hızlandırıcı donanım test planı.....	58
Tablo 5.15: Lenet modeli iki farklı katlama konfigürasyonu.....	58
Tablo 5.16: Lenet hızlandırıcı donanımları kaynak kullanımı ve verim tablosu.....	59
Tablo 5.17: Lenet modeli hızlandırıcılar Vivado ile hesaplanan güç tüketimleri.....	61
Tablo 5.18: Evrişim ve tam bağlantılı katmanlar için hızlandırıcı katlama konfigürasyonu.....	63
Tablo 5.19: Tasarlanan Lenet hızlandırıcılarının kaynak verim tablosu.....	64
Tablo 5.20: Normalleştirme katsayıları.....	64
Tablo 6.1: Geliştirilen FPGA tabanlı derin öğrenme hızlandırıcıları özellik tablosu.....	69
Tablo 6.2: Benzer çalışmaların sonuçlarının karşılaştırılması.....	70

SEMBOL LİSTESİ

S	: Öznitelik matrisi
I	: Giriş matrisi
K	: Filtre matrisi
i, j	: Giriş matrisi indisleri
m, n	: Filtre matrisi indisleri
Qnt_{th}	: Niceleme maksimum değer
N	: Niceleme bit genişliği
th	: Niceleme sonucunun en yüksek mutlak değeri
k	: Ölçek
QntA	: Nicelenmiş A matrisi
NHWC	: Grup, yükseklik, genişlik ve kanal sayısı
F	: Katlama faktörü
P	: İşlem sayısı
F_n	: Nöron katlama
F_s	: Sinaps katlama
T_f	: Toplam katlama
L_{fps}	: Katman saniyedeki işlenen görüntü sayısı
F_{clk}	: Çalışma frekansı (Hz)
GOP_s	: Saniyede yapılan milyar işlem sayısı
TOP_s	: Saniyede yapılan trilyon işlem sayısı
GFLOP_s	: Saniyede yapılan milyar kayar noktalı işlem sayısı
GOP_{s/w}	: Watt başına saniyede yapılan milyar işlem sayısı

KISALTMALAR LİSTESİ

A	: Aktivasyon niceleme bit genişliği
ASIC	: Uygulamaya özel entegre devreler (Application specific integrated circuit)
AXI	: Gelişmiş genişletilebilir arayüz (Advanced extensible interface)
BNN	: İkileştirilmiş sinir ağları (Binarized neural networks)
BRAM	: Blok rastgele erişimli bellek (Blok random access memory)
CNN	: Evrişimsel sinir ağları (Convolutinonal neural networks)
CNV	: Evrişim katmanı
DIV	: Bölme bloğu
DNN	: Derin sinir ağları (Deep neural networks)
DSP	: Dijital sinyal işleme birimi (Digital signal processing)
FC	: Tam bağlantılı katman (Fully connected layer)
FF	: Flip flop kaynakları
FPGA	: Alan etkili kapı dizileri (Field programmable gate arrays)
GPU	: Grafik işlemci ünitesi (Graphics processing unit)
H	: Yüksek katlama seviyesi
HLS	: Yüksek seviyeli sentez (High level synthesis)
I	: Giriş niceleme bit genişliği
IP	: Fikri mülkiyet (Intellectual Property)
L	: Düşük katlama seviyesi
LSTM	: Uzun kısa süreli bellek (Long short-term memory)
LUT	: FPGA mantık blokları
LUTRAM	: LUT bellekleri
M	: Orta katlama seviyesi
MAC	: FPGA çarpma toplama ve çarpma-biriktirme blokları (FPGA Multiplication, addition and multiply-accumulate)
MLP	: Çok katmanlı sinir ağı modeli (Multi layer perception)
MUL	: Çarpma bloğu
MUX	: Seçici (Multiplexer)
MVTU	: Matris vektör eşikleme birimi (Matrix vector threhsold unit)
NN	: Sinir ağları (Neural networks)
ONNX	: Açık sinir ağları değişimi (Open neural networks exchange)
PE	: İşleme elemanları (Processing elements)
PL	: Programlanabilir mantık (Programmable logic)
PTQ	: Eğitim sonrası niceleme (Post training quantization)
PS	: İşleme sistemi (Processing system)
PS7	: ZYNQ işletim sistemi
PT	: İşleme ızgarası (Processing tile)
QAT	: Eğitim esnasında niceleme (Quantization avare training)
QNN	: Nicelleştirilmiş sinir ağları (Quantized neural networks)
RAM	: Rastgele erişimli bellek (Random access memory)
RTL	: Kayıt aktarım düzeyi (Register-transfer level)

SIMD	: Çoklu veri akışı giriş sayısı
STE	: Doğrudan tahmin edici (Straight through estimator)
SUB	: Çıkarma bloğu
SVM	: Destek vektör makineleri (Support vector machines)
SWU	: Kayar pencere birimi (Sliding window unit)
W	: Ağırlık niceleme bit genişliği
YSA	: Yapay sinir ağırları

ÖNSÖZ

Bu tezde derin öğrenme ağlarının taşınabilir platformlarda uygulanabilmesi için gerekli olan FPGA tabanlı hızlandırıcılar üzerinde çalışılmıştır. Derin öğrenme hızlandırıcıları ile ilgili çalışmalar özellikle son beş yılda artmış olmasına karşın bu alanda yeterli çalışma yapılmaması karşılaştığımız en büyük zorluklardan birisidir. Çalışmalara öncelikle RTL seviyesinde tasarım ile başlandı, ardından tasarım karmaşıklığı ve tasarım sürecinin her model için yeniden başlaması nedeni ile HLS tasarıma geçildi. HLS seviyesinde hızlandırıcı tasarlandı ancak nicelenmiş ağlarda yaşadığımız tasarım problemi nedeni ile tezin bu aşamasından sonra hızlandırıcı tasarımı için Xilinx araştırma ekibi tarafından geliştirilen FINN çerçevesi kullanıldı.

FINN çerçevesi geliştiricilerine ve özellikle Yaman Umuroğlu'na, FINN eğitimi FPGA'21 konferansına katılımımız ve sonraki süreçte sağladıkları katkılar adına teşekkürlerimi sunarım. Doktora eğitimim boyunca, akademik bilgileri ve tecrübelerini benden esirgemeyen bu tezin planlanmasında ve yürütülmesinde büyük katkıları olan değerli danışman hocam Prof. Dr. Ayhan İstanbullu'ya, tavsiye ve eleştirileri ile tezin gelişimine katkı sunan tez izleme komitesi üyeleri Prof. Dr. Metin Demirtaş ve Doç. Dr. Metin Bilgin hocalarıma teşekkürü bir borç bilirim. Lisans üstü eğitim hayatım boyunca desteklerini hep hissettiğim eşim Hülya Ayhan Taşcı'ya, bu zorlu süreçte babalarını sabır ve fedakarlıkla destekleyen çocuklarım Ahmet Selim, Azra ve Neva'ya sevgi ve teşekkürlerimi sunarım.

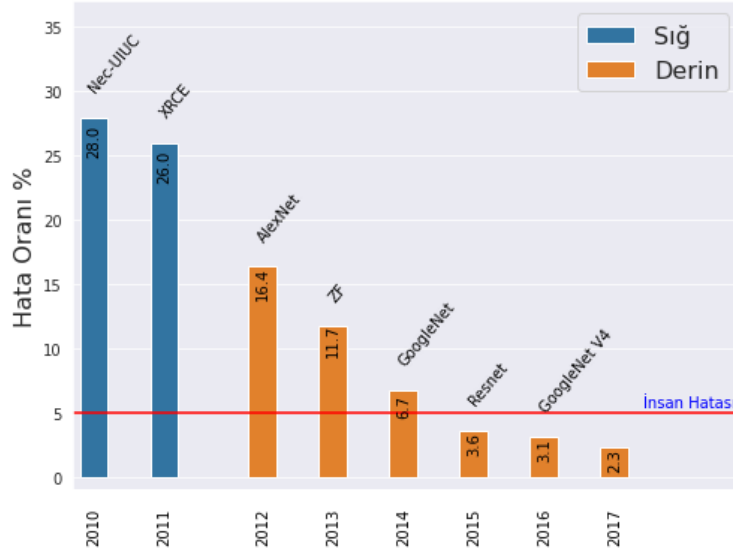
Balıkesir, 2022

Mustafa TAŞCI

1. GİRİŞ

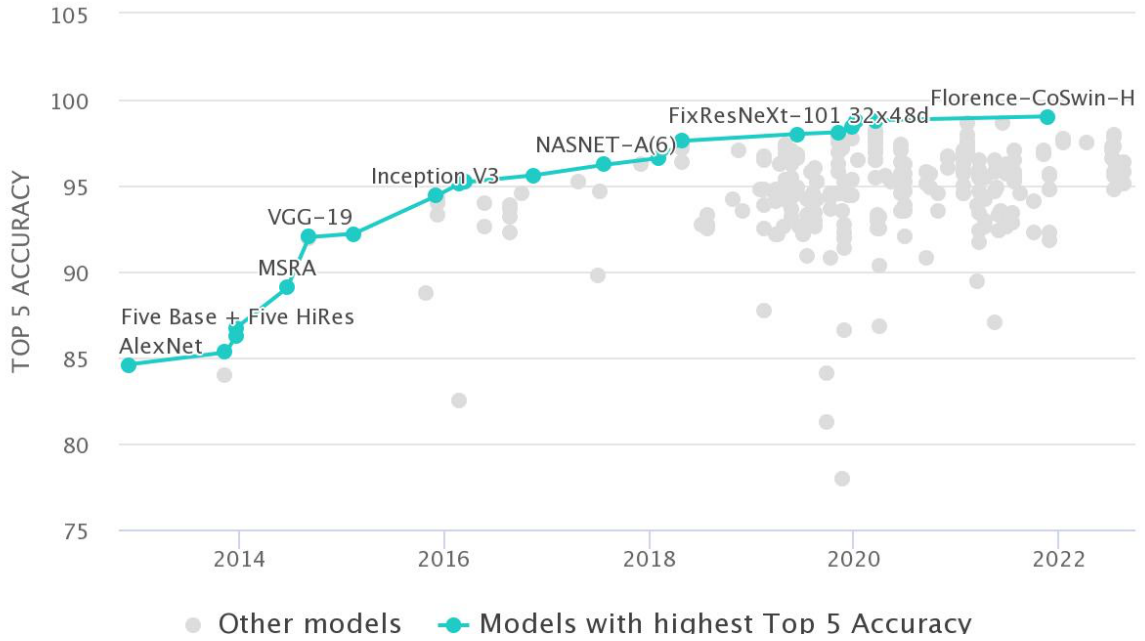
İnsan hayatını kolaylaştırmak adına geliştirilen yapay zekâ barındıran akıllı makineler 21. yy. başından bu yana teknolojik gelişmelerin odağı haline gelmiştir. Ev sakinlerinin alışkanlıklarına göre ısıtılan ya da soğutulan akıllı evler, sürücüsüz hareket eden otomobiller, sosyal mesajlaşma platformlarını analiz edip ürün geliştiren akıllı sistemler bu yeni teknolojinin insanoğluna sunduğu ürünlerden sadece birkaçıdır. Günümüzde pek çok alanda karşımıza çıkan yapay zekâ uygulamalarında özellikle derin öğrenme teknolojisi oldukça yaygın kullanılmaktadır. Yaygın olarak kullanılan en önemli yapay zekâ teknikleri yapay sinir ağları, makine öğrenmesi ve derin öğrenmedir. 1950'li yılların başlarında Yapay sinir ağları ve karar ağaçları bilgisayarlı görü için kullanılan iki yöntemdi. Yapay sinir ağları insan beyninin çalışma mekanizmasını taklit eden çok katmanlı bir ağ yapısıdır [1]. Makine öğrenimi, çevredeki ortamdan algıladıkları ya da veri setlerinden öğrenerek gelişen yapay zekâ dalıdır [2]. Son yıllarda internet ve özellikle mobil telefonlarda yaşanan teknolojik ilerleme, resim, video ve ses dosyaları gibi dijital verilerin hızla çoğalarak büyük veriyi ortaya çıkarmalarına sebep oldu. Derin öğrenme, sinir ağları araştırmaları alanında uzun bir aradan sonra 2006'da ortaya çıktı. Derin öğrenmede ağların öznitelikleri ve ağırlıkları makine öğrenmesinden farklı olarak insanlar tarafından belirlenmez. Bunun yerine ağırlık ve ağ, genel amaçlı bir öğrenme prosedürü kullanılarak büyük veri setlerinden öğrenilirler [3]. Derin öğrenme endüstriyel ürünler başta olmak üzere, tıp, güvenlik, iktisat, otonom araçlar ve robotik gibi pek çok alanda kullanılmaktadır.

Evrişimsel sinir ağları (CNN), biyolojik görme sistemlerinden ilham alınarak, yapay sinir ağları (YSA) ile geliştirilen bir derin öğrenme algoritmasıdır [4]. 2012 yılında düzenlenen ImageNet büyük ölçekli görüntü tanıma yarışmasını CNN modeli ile tasarlanan ağ projesi kazanmıştır. Bu sonuç derin öğrenme tekniğine olan ilgiyi artırmış ve bu alanda yapılan çalışmaların hız kazanmasına sebep olmuştur. Bu yarışmada Top-5 hata oranı %26,1'den %16,4 seviyelerine düşürülmüştür.



Şekil 1.1: ImageNet yarışması sonuçları.

Son yıllarda yapılan çalışmalarla ImageNet hata oranı insan hata oranının altına düşerek %3 seviyelerine ulaşmıştır. 2021 yılında Yuan L. ve arkadaşları yaptıkları çalışma ile ImageNet 1-K TOP-5 seviyesinde %97,18 başarı raporladılar [5]. Yapay görü alanında insan performansının üstünde doğruluk oranı elde edilmesi derin öğrenme modellerinden CNN algoritmalarının yaygınlaşmasına sebep olmuştur.



Şekil 1.2: ImageNet 2022 yılına kadar yapılan derin öğrenme projeleri TOP-5 doğruluk oranları.

Derin öğrenmede kullanılan algoritmalar dijital verilere ihtiyaç duyarlar. Günümüz teknolojisinde yaygınlaşan dijital fotoğraf makineleri ve cep telefonları ile çekilen fotoğraf ve video görüntüleri sosyal ağlara ve arama motorlarına yüklenmektedir. Bu görüntüler derin öğrenme uygulamaları için uygun bir veri kümesi haline getirilmektedir. Hızla artan bu veri kümeleri son yıllarda "Büyük Veri (Big Data)" olarak adlandırılmakla birlikte makine öğrenimi ve derin öğrenme uygulamalarının gelişmesine olanak sağlamıştır [6].

2. FPGA TABANLI DERİN ÖĞRENME HIZLANDIRICILARI

Hızla gelişen derin öğrenme modellerinde gizli katmanların sayıları artırılmaktadır. Bu durum daha büyük belleklere ve yüksek işlemci hızlarına ihtiyaç duyulmasını ortaya çıkarır. Derin öğrenmede kullanılacak görüntü 640x480x3 boyutta ve 5000 adet ise, sistemin eğitimi için ihtiyaç duyulan hafıza miktarı 640x480x3x5000 byte olur. CNN algoritmaları yoğun matematiksel işlemler içerdiğinden, bir derin öğrenme ağının eğitilmesi ağın derinliğine ve parametrelere bağlı olarak günlerce sürebilir. Bir derin öğrenme modelinin eğitilmesi esnasında geriye yayılım algoritmasında kullanılan hesaplamalar paralel işlemciler ile CPU'lardan daha hızlı gerçekleştirilebilir. Bu nedenle derin öğrenme ağlarında özellikle eğitim aşaması için işlemci yerine grafik işlemciler (GPU) kullanılmaktadır. Her ne kadar GPU derin öğrenmenin hesaplama etkinliğinde üstün performans gösterse de pahalıdır ve yüksek güç tüketimine sahiptir. Aynı işlevsel tasarım için, tek bir GPU'nun güç tüketimi genellikle FPGA'nın güç tüketiminin onlarca, hatta yüzlerce katıdır. FPGA'lerin yeniden yapılandırılabilme, özelleştirilebilme, enerji verimliliği ve esnek mimari tasarımı gibi avantajları nedeniyle, gittikçe daha fazla sayıda araştırmacı FPGA tabanlı CNN donanımı uygulamasına odaklanmaktadır [7].

CONV CoProcessor, Sankaradas ve arkadaşları tarafından, Virtex5 LX330T FPGA platformu kullanarak yapılmış bir paralel yardımcı işlemci çalışmasıdır [8]. Çalışmanın amacı CNN'leri hızlandırmak için paralellliği kullanarak bir hesaplama motorunu geliştirmektir. Yardımcı işlemci prototipi, CNN ileri yayılımı için saniyede 3,4 milyar çarpma işlemi gerçekleştirebilmektedir. Bu, 2.2 GHz AMD Opteron işlemcideki bir yazılım uygulamasından 31 kat daha hızlıdır. Çalışmada yüz tanıma uygulamasının normal CPU ya göre 6-10 kat arasında daha hızlı gerçekleştirilebileceği raporlanmıştır.

Cadambi S. ve arkadaşları yapılandırılmamış büyük miktarda veri içeren uygulamalarda hem öğrenme hem de sınıflandırma görevlerini hızlandırmak için MAPLE donanımını sundu [9]. MAPPLE, beş farklı iş yükü (SVM, SSI, GLVQ, K-means ve CNN) için 125MHz'de çalışan Virtex5 SX240T FPGA üzerinde geliştirilebilir bir hızlandırıcıdır. Bu uygulamalar için vektör ya da matris işlemlerini ızgaralar haline dönüştürerek matris çarpımlarını verimli bir şekilde gerçekleştirdiler. MAPLE, bellek içi işlemeyi kullanarak büyük ara verileri anında işlemek için çip üzerinde akıllı bellek blokları kullanır. CNN ağı ile yüz tanıma

uygulamasında, Intel xeon quad-core 2.5 GHz CPU'dan yaklaşık 3 kat, 2.3 GHz NVIDIA Tesla C870 GPU'dan ise %50 daha hızlı olduğu tespit edilmiştir.

Chakradhar ve arkadaşları [10], tarafından geliştirilen DC-CNN, FPGA üzerinde dinamik olarak konfigüre edilebilen bir CNN önişlemci uygulamasıdır. Daha önceki çalışmalarda CNN'lerde hızlandırıcı uygulamaları yapılmış özellikle evrişim işlemlerinden kaynaklanan iş yüklerinin paralelleştirildiği görülmüştü. Bu paralelleştirme işlemi standart modellere ve iş yüklerine özeldi. Bu çalışmada ise iş yüklerini otomatik olarak analiz ederek, farklı türde donanım ve yazılım bileşenlerinin karışımına uyacak şekilde dinamik olarak yapılandırılan bir işlemci mimarisi oluşturmaya çalışılmıştır. Önerilen sistem Otomobil Güvenliği, Yüz Tanıma, Mobil Robot Görüsü vb. beş farklı CNN uygulamasında test edilmiş ve Intel Xeon 8 çekirdekli işlemci ve 1.35GHz 128 çekirdekli NVIDIA GPU ile hız ve güç tüketimleri açısından karşılaştırılmıştır. Yüz tanıma uygulamasında CPU'dan 4 kat daha hızlı olduğu raporlanmıştır. Önerilen DC-CNN yardımcı işlemci, saniyede 25 ile 30 kareyi kolayca işleyerek, çok çeşitli nesne algılama ve tanıma görevlerinde gerçek zamanlı video akışı işlemeyi sağlayan ilk CNN mimarisidir. Ayrıca bu sistem 14 W enerji tüketirken GPU 150 W'tan fazla enerji tüketmektedir.

Farabet ve arkadaşları daha önce tasarlamış oldukları CNP mimarilerini geliştirerek neuFlow isimli bu çalışmayı sundular [11]. Çalışmada genel amaçlı görüş algoritmaları için optimize edilmiş bir veri akışı donanım mimarisi olan neuFlow ve bu algoritmaların yüksek seviyeli akış grafiği temsillerini makine koduna dönüştüren bir veri akışı derleyici luaFlow geliştirilmiştir. NeuFlow, aritmetik mantık birimlerinin 2B ızgaralarını 2B işleme karoları (PT) ile değiştirmiştir. Bu mimari, bir 2B PT ızgarası, bir kontrol ünitesi ve bir doğrudan erişimli bellek modülü içerir. PT blokları bir seçici (MUX), çarpma, toplama vb. işlem operatörlerinden meydana gelir. Bu bloklar evrişim ve havuzlama gibi işlemleri gerçekleştirmektedir. Sunulan çalışmada Xilinx Virtex6 VLX240T FPGA platformu kullanılmış, sistemin 10 W enerji tükettiği belirtilmiştir. Bu bir dizüstü bilgisayardan 10 kat daha düşük enerji tüketimi anlamına gelmektedir. NeuFlow V6 FPGA üzerinde yapılan CNN uygulamasında saniyede 147 GOPs yapabilirken Intel DuoCore 2.7 GHz CPU 1.1 GOPs işlem yapabilmektedir.

Peemen ve arkadaşları [12] CNN uygulamalarında sınırlı miktarda harici bellek bant genişliği problemini çözmeye odaklandılar. Bu çalışmada esnek bir bellek hiyerarşisi ile

bellek darboğazının etkilerinin azaltılabileceği gösterilmiştir. Çip üzerindeki bellek boyutunun en aza indirilmesini sağlayarak alan ve enerji kullanımını azaltır. Bellek merkezli hızlandırıcı, blok RAM (BRAM) tabanlı çok banklı onchip arabelleklerini kullanmaktadır. Hızlandırıcı evrişim katmanını hızlandırmak için esnek yeniden kullanım arabelleklerine sahip bir SIMD MAC PE kümesi kullanır. Çalışma Virtex6 FPGA'lar üzerinde geliştirildi. Standart belleklere sahip hızlandırıcı ile aynı performansı 13 kata kadar daha az FPGA kaynağı kullanarak gerçekleştirebilmektedir. Aynı miktarda FPGA kaynağı kullanıldığında hızlandırıcı 11 kata kadar daha hızlıdır.

Neural network next (nn-X) [13] derin sinir ağlarının gerçek zamanlı yürütülmesini sağlamak için ölçeklenebilir, düşük güçlü bir yardımcı işlemcidir. Derin öğrenme ağlarında sıkça kullanılan evrişim, alt örnekleme vb. işlemleri koleksiyon adını verdikleri bir dizi yapılandırılabilir ögede tutmaktadırlar. nn-X'in mimarisi sistemi yöneten bir ana bilgisayar işlemcisi SoC, bir yardımcı işlemci ve harici bellekten oluşur. nn-X sistemi, DDR3 belleğe 4 yüksek hızlı doğrudan bellek erişim arabirimi ve iki ARM Cortex-A9 işlemci ve Zynq XC7Z045 FPGA platformu içerir. Yardımcı işlemci, içinde evrişim, havuzlama ve doğrusal olmayan operatörler içeren koleksiyonları kullanarak işlemleri paralel hale getirerek öğrenme ağlarını hızlandırır. nn-X, yapılan testlerde 200 GOPs'lik performansa ulaşabilirken, 4 W'tan daha az güç tüketmektedir. Ayrıca yüz tanıma uygulaması performans testlerinde üzerinde bulunan ARM Cortex-A9 işlemciden 115 kat daha hızlıdır.

Zhang ve arkadaşları [14] CNN ağlarında hesaplama çıktısının, bir FPGA platformu tarafından sağlanan bellek bant genişliği ile iyi eşleşmeyebilmesi problemini çözmek amacıyla çatı hattı tabanlı bir tasarım şeması sundular. Çalışmalarındaki odak noktası, tahmin süreci sırasında hesaplama süresinin %90'ından fazlasını tükettiği için öncelikle evrişim katmanlarını hızlandırmaktır [15]. Bunu yaparken, yazarlar hem hesaplama işlemlerini hem de evrişim katmanlarında bellek erişim işlemlerini optimize ettiler. Tavan hattı performans modeli, tasarım alanındaki tüm olası çözümlerden en uygun tasarımı belirlemek için kullanılır. Çalışma Xilinx VC707 FPGA kartına uygulandı. Uygulama sonuçları CPU ya göre 17,42 kat daha hızlı olduğunu ve sadece 18,6 W enerji tüketerek CPU tabanlı sistemden 24,6 kat daha fazla enerji verimli olduğunu göstermektedir.

Microsoft araştırma ekibi 2014 yılında Catapult projesini duyurdu. Bu projede, veri merkezindeki FPGA'ları kullanarak Bing sıralamasını yaklaşık 2 kat hızlandırmayı

amaçlanmıştır. Araştırma ekibi, sunucu gücünün küçük bir bölümünü tüketirken yüksek performans elde eden yüksek verimli FPGA tabanlı bir CNN hızlandırıcısı geliştirdi. 2015 yılında Ovtcharov ve arkadaşları [16] Microsoft araştırma merkezinde CNN ağlarını eğitim aşamasında hızlandırmak için Starix-V GSMD5 FPGA platformu ile Catapult donanımını kullandı. Çalışma ImageNet-1K veri setini AlexNet modeli üzerinde 25 W enerji tüketerek saniyede 134 görüntüyü sınıflandırabilmiştir. Sunulan çalışma Zhang ve arkadaşlarının [17] yapmış olduğu çalışmanın sonuçlarından üç kat daha iyi bir performans sergilemiştir. Arria 10 GX 1150 FPGA kullanılırken verimin 233 görüntü/s'ye yükseldiği belirtilmiştir. Enerji tüketimi açısından incelendiğinde Tesla K40 GPU dan yaklaşık dokuz kat daha düşük enerji kullanmıştır.

Qiu ve arkadaşları [18] imageNet büyük ölçekli bir görüntü sınıflandırması için FPGA tabanlı bir CNN hızlandırıcı tasarımı önerdi. Çalışmada evrimsel katmanların hesaplama merkezli ve tam bağlantılı katmanların bellek merkezli olduğu vurgulanarak, bu iki farklı katman için iki ayrı çözüm üzerinde durdular. Çalışmada bant genişliğini ve kaynak kullanımını iyileştirmek için CNN'deki tüm katman türleri için verimli olan dinamik hassas veri niceleme yöntemi ve konvolver tasarımı önerilmiştir. Niceleme işlemi VGG16 modelinde 8/4 bit niceleme ile test edilmiş, doğruluk kaybının kayar noktalı uygulamaya göre %0,4 gibi çok düşük bir değer olduğu tespit edilmiştir. Ayrıca harici bellek kullanımında bant genişliğinin daha yüksek kullanılabilmesi için bir veri düzenleme yöntemi geliştirilmiştir. FPGA üzerindeki FC katmanlarının performansı, bant genişliği ile sınırlı olmasına rağmen ARM işlemcilerden daha yüksektir. Geliştirilen hızlandırıcının 150 MHz çalışma frekansında 136,97 GOPs işlem kapasitesine ve %86,66 Top-5 doğruluk oranına sahip olduğu tespit edilmiştir. GPU performansı FPGA'ye göre 13 kat daha iyi olmasına karşın, GPU 26 kat daha fazla güç tüketmektedir.

Derin öğrenme modellerinin tasarım parametreleri uygulamadan uygulamaya önemli ölçüde farklılık göstermektedir. Bu nedenle, genel amaçlı bir FPGA hızlandırıcı tasarlamak zordur. DeepBurning [19] bu sorunu çözmek için genel amaçlı FPGA tabanlı bir sinir ağı tasarım otomasyon aracıdır. DeepBurning, kullanıcı tarafından belirlenen kısıtlamalar ile RTL seviyesinde kontrol akışını ve veri düzenini oluşturan bir hızlandırıcı oluşturmaya odaklanır. Yazarlar DeepBurning'in performansını Zhang ve arkadaşları [14] geliştirmiş olduğu hızlandırıcı ile AlexNet CNN modeli üzerinde karşılaştırdı. DeepBurning'in 1,45 kat daha az enerji tüketmesine karşın 1,13 kat daha yavaş olduğu görülmüştür. Suda N. ve arkadaşları

büyük ölçekli evrişimli sinir ağı modellerini hızlandırmak için OpenCL tabanlı bir optimizasyon çerçevesi önerdi [20]. Zhang ve arkadaşları [17] tarafından sunulan Caffeine, Caffe derin öğrenme çerçevesinden model dosyalarını ve FPGA cihaz özelliklerini kullanarak en iyi donanım parametrelerini otomatik olarak seçer. Caffeine FPGA donanımını oluşturmak için HLS tabanlı bir mimari kullanır. FpgaConvNet [21] hedeflenen FPGA platformunun özelliklerine göre, kullanılacak olan CNN hızlandırıcıyı HLS komut dosyalarını kullanarak oluşturur. FpgaConvNet, katlama faktörlerini ve bölümlene noktalarını en verimli hale getirmek için benzetilmiş tavlama (simulated annealing) kullanır. FpgaConvNet Zynq-7000 XC7Z7020 FPGA platformu üzerinde LeNet-5 modeli ile test edilmiş ve CNP'den 1,62 kat daha iyi performans sağladığı görülmüştür. Liu ve arkadaşları [22] FPGA tabanlı CNN hızlandırıcıları için paralelleştirme işlemini dört ayrı seviyeye ayırdı. Bu seviyeler görev seviyesi, katman seviyesi, döngü seviyesi ve operatör seviyesidir. Yazarlar dört paralellik seviyesinden yararlanan bir paralel çerçeveyi sundular.

FP-DNN [23] TensorFlow'dan DNN'lerin sembolik açıklamalarını girdi olarak alan ve modele karşılık gelen FPGA tabanlı hızlandırıcıyı otomatik olarak oluşturan uçtan uca bir çerçevedir. RTL-HLS hibrit şablonları kullanılarak, modeli oluşturan matris çarpımı gibi genel amaçlı hesaplamalar hızlandırıcılara dönüştürülmektedir. Sunulan çerçeve CNN, LSTM, RNN ve Artık Ağlar gibi tüm DNN türlerini desteklemektedir. Bu çalışma ResNet-152 modelini FPGA üzerinde uygulayan ilk çalışmadır. Çalışma Stratix-V GSMD5 FPGA üzerinde VGG-19, LSTM-LM, ResNet-152 DNN modelleri ile test edilmiştir. FP-DNN 2 adet 2,6 GHz 8 çekirdekli Intel Xeon E5-2650v2 işlemci içeren sunucudan yaklaşık 3,06 kat daha hızlı çalışmıştır. Mobil cihazlar gibi düşük hafıza birimi ve işlemci gücüne sahip cihazlarda derin öğrenme modellerinin kullanılabilmesi ağırlık, aktivasyon ve giriş bit genişliklerinde nicelendirme ihtiyacını doğurdu. Bu alanda Rastegari M. [24] xnet'ini, Kim M. [25] Bitwise Neural Network'ü, Zhou S. [26] DoReFa-net'i, Courbariaux M. [27] Binarized Neural Network isimli çalışmaları sundular. Bu çalışmaların ortak noktası CNN modellerinin nicelenmesi için ağ modelleri önermeleridir. Tezin nicelendirme kısmında bu çalışmalar daha detaylı işlenecektir.

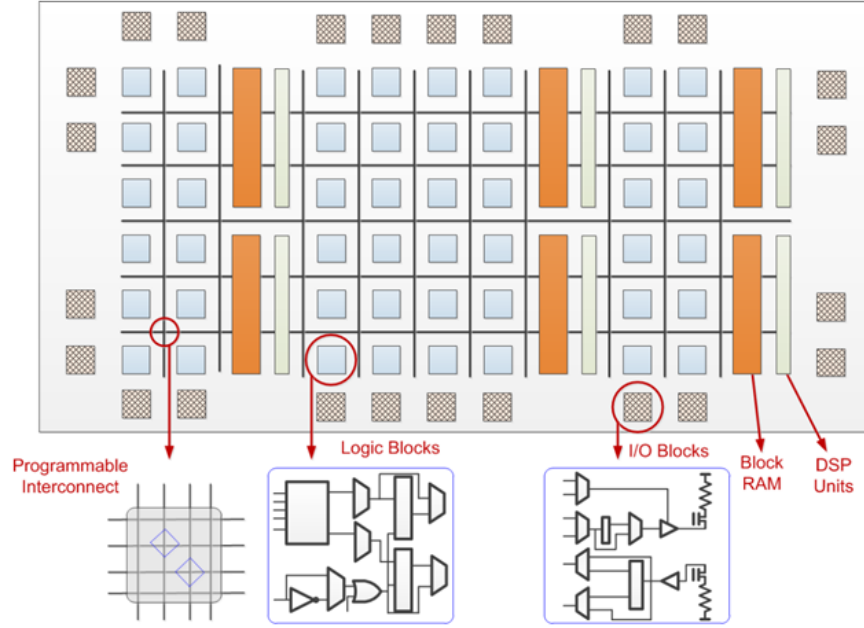
Umuroğlu ve arkadaşları [28] 1 bitlik ya da 2 bitlik nicelendirmelerle çatı hattı modeli üzerinde çalışmalar yaptılar. Bu çalışmalarda MNIST veri seti üzerinde 8 bitlik bir CNN modeline göre 2-11 kat daha fazla parametre ve işlem kullanılması gerektiğini buldular. Çalışmada BNN ağların CNN ağlara kıyasla 16 kat daha hızlı olduğunu belirttiler. Yazarlar önceden

eđitilmiş bir BNN ađını FPGA üzerinde uçtan uca oluřturan FINN çerçevesini sundular. Bu çalıřmada BNN'i oluřturan tüm düđüm noktaları C++ ve HLS kodları ile otomatik oluřturulmaktadır. Sunulan çalıřmanın en önemli özelliklerinden biri evriřim katmalarında toplama yerine bit sayacı (popcount) kullanılmasıdır. Bu FPGA kaynak kullanımında LUT ve Flip Flop kullanım sayısını yarı yarıya azaltmaktadır. Hızlandırıcı mimarisi bir kayan pencere birimi (SWU) ve matris vektör biriminden (MVTU) oluřmaktadır. Evriřim iřlemi MVTU biriminde gerçekteřmektedir. PE, SIMD ve FIFO parametreleri kullanılarak katlama oranları belirlenebilmektedir. Katlama iřlemi ađın hızını iyileřtirirken, kaynak kullanımını artırmaktadır. FINN SoC özellikli Xilinx FPGA modellerini desteklemektedir. Sunulan çalıřma CIFAR-10 veri seti ile VGG16 benzeri bir CNN mimarisi üzerinde test edilmiřtir. Güç tüketimi 11,7 W olurken, 2.5 TOPs iřlem performansı sergilemiřtir.

Aydonat ve arkadařları [29] , verilerin yeniden kullanımını ile harici bellek bant geniřliđini önemli ölçüde azaltan OpenCl tabanlı bir hızlandırıcı mimarisi sundular. Çarpma biriktirme iřlemlerinde Winograd dönüřümünü kullanarak iřlem sayısını %50 azaltmayı bařardılar. DLA mimarisi Arria 10 FPGA kitine AlexNet CNN modeli ile uygulanarak deđerlendirildi ve 1020 görüntü/s'lik bir performans elde edildi. Önceki yaklařımlarda FPGA tabanlı hızlandırıcılar CNN katmanlarını birer birer hesaplayan tek bir iřlemci oluřturmaktadırlar. Tasarlanan iřlemci boyutları deđiřen farklı CNN katmanlarını hesaplamak için kullanılmaktadır. Shen ve arkadařları [30], FPGA kaynaklarını CNN'in her bir evriřim katmanı için uyarlanmış birden çok iřlemciye bölen bir otomatik tasarım metodolojisi sundular. Sunulan çalıřma Xilinx Virtex-7 FPGA üzerinde AlexNet uygulaması ile test edildi benzer uygulamalardan 3,8 kat daha hızlı olduđu belirtildi. ALAMO [31] hızlandırılacak derin öđrenme modelinin algoritma yapısını ve parametrelerini analiz eden ve bir FPGA üzerinde çeřitli derin öđrenme algoritmalarını otomatik olarak entegre eden bir derleyicidir.

3. ALANDA PROGRAMLANABİLİR KAPI DİZİLERİ (FPGA)

Alanda programlanabilir kapı dizileri (FPGA), programlanabilen mantık blokları ve bu blokları birbirine bağlayan bağlantılardan meydana gelen lojik devrelerdir. FPGA içinde barındırdığı lojik kapı, FF vb. devre elemanlarının tasarımcının yazmış olduğu kodlara göre birbirine bağlanması ile ortaya çıkan tmleik devrelerdir.



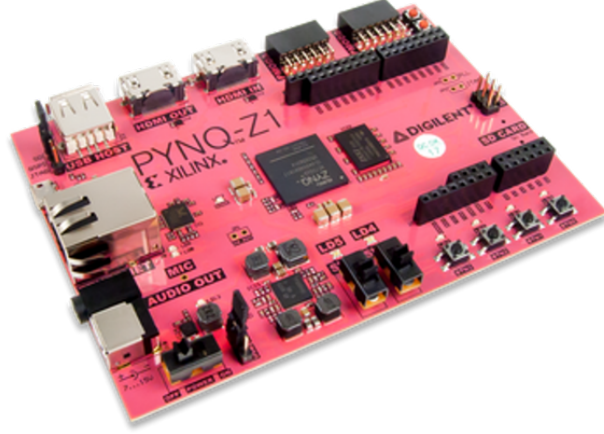
ekil 3.1: FPGA blok diyagramı [61].

3.1 PYNQ Yazılımı

PYNQ, Xilinx FPGA platformlarının kullanımını kolaylatıran Xilinx'in aık kaynaklı bir projesidir. PYNQ, gml sistem tasarımcılarının, programlanabilir mantık devreleri tasarlamak iin ASIC tarzı tasarım aralarını kullanmak zorunda kalmadan FPGA platformlarını kullanmalarını saęlar. PYNQ, adından da anlaılacaęı gibi, "Python Productivity for Zynq" anlamına gelir. Popler bir programlama dili olan Python yardımıyla Zynq cihazları kullanılarak gml uygulamalar oluturma srecini basitletirir, gml sistem gelitiriciler iin harika bir aratır. PYNQ ile video ileme (yksek kare hızlı), paralel donanım yrtme, gerek zamanlı sinyal ileme, donanım hızlandırılmalı algoritmalar, vb. yksek performanslı uygulamalar gelitirilebilmektedir. PYNQ yazılımı, Zynq, Zynq UltraScale+, Zynq RFSoc, Alveo hızlandırıcı kartları ve AWS-F1 gibi donanımlar ile birlikte kullanılabilir.

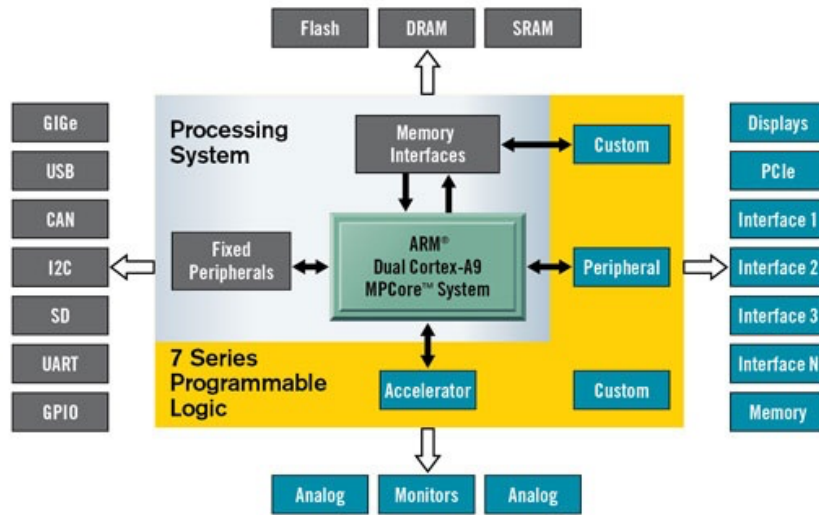
3.2 Xilinx PYNQ Z1 FPGA Platformu

PYNQ-Z1 kartı, programcılarının yerleşik işlemciyi (SoC) Python ile programlamasını sağlayan PYNQ açık kaynak çerçevesiyle birlikte kullanılmak üzere tasarlanmıştır. Bir FPGA'nın programlanmasını çift çekirdekli bir ARM Cortex -A9 işlemci ile birleştiren Xilinx Zynq -7000 SoC etrafında tasarlanmıştır.



Şekil 3.2: Xilinx PYNQ Z1 FPGA kartı.

Zynq işletim sistemi, (PS) olarak adlandırılan çift çekirdekli ARM Cortex-A9 işlemciye ve programlanabilir mantık (PL) olarak adlandırılan FPGA yapısıyla bütünleşmiş bir çip üzerinde sistemdir (SOC).



Şekil 3.3: PYNQ Z1 Blok diyagramı.

4. DERİN ÖĞRENME

Derin öğrenme, büyük miktarda veriden öğrenen, üç ya da daha fazla katmandan meydana gelen sinir ağlarıdır. Derin öğrenmeyi makine öğrenmesinden ayıran en önemli özelliği çok yüksek miktarda veriyi kullanarak öznitelikleri otomatik olarak belirlemesi ve eğitimde kullanmasıdır.

4.1 Derin Öğrenme Çerçeveleri

Derin öğrenme algoritmalarının uygulanabilmesi için çok çeşitli çerçeveler geliştirilmiştir. Bu çerçeveler bir derin öğrenme modelini tasarlamayı ve derin öğrenme için kullanılan işlemlerin gerçekleştirilmesi işlevlerini kolaylaştıran kütüphanelerdir. Mevcut derin öğrenme çerçeveleri incelenmiş, Tablo 4.1’de genel hatlarıyla özetlenmiştir.

Tablo 4.1: Derin öğrenme çerçeveleri karşılaştırma tablosu.

Çerçeve	Geliştirici	Platform	Ara yüz	Paralleleştirme
Theano	Montreal Üni.	CrossPlatform	Python	OpenMP, CUDA
DL4j	SkyMind E. T.	Crossplatform	Java, Scala, Clojure	OpenMP, CUDA
Caffe	Berkeley L. C.	Ubuntu, OSX, AWS	C++, Python	OpenMP, CUDA
PyTorch	Facebook	Linux, MacOS, Windows	C++, Python	OpenMP, CUDA
TensorFlow	Google Brain T.	Linux, MacOS,	C++, Python	CUDA
Keras	François Chollet	Linux, MacOS, Windows	Python, R	CUDA

Bu tez çalışmasının ilk uygulamasında CAFFE çerçevesi kullanılırken ikinci uygulamada Brevitas niceleme kütüphanesinin işlevselliği nedeniyle PyTorch çerçevesi kullanılmıştır.

4.2 Derin Öğrenme Veri Setleri

4.2.1 Mnist Veri Seti

El yazısı, rakam tanıma ve optik karakter tanıma önemli bir sorundur. Bu problem yıllardır makine öğrenme ve derin öğrenme algoritmaları için bir test örneği olarak kullanılmaktadır. MNIST (Modified National Institute of Standards and Technology); görüntü işleme algoritmalarının eğitimi ve testi için yaygın olarak kullanılan büyük bir el yazısı rakam veri setidir. MNIST veri seti, Amerika standart enstitüsü (NIST) tarafından oluşturulan orijinal veri seti düzenlenerek geliştirilmiştir. Bu veri setinde eğitim için 60.000, test için ise 10.000

el yazısı rakam görüntüsü vardır. Eğitim setindeki görüntüler Amerikan Sayım Bürosu çalışanlarından, test setindeki görüntüler ise Amerikan Lise öğrencilerinden alınmıştır. NIST veri setindeki tüm siyah beyaz rakamlar 28x28 piksel gri tonlu resimlere dönüştürülmüştür. Bu dönüşüm esnasında resimlerin ağırlıkları resmin ortasında tutulmuştur. Bu dönüşümler sonucu Y. Lecun tarafından oluşturulan MNIST veri seti, 2011 yılında <http://yann.lecun.com/exdb/mnist/> adresli web sayfasında yayınlanmıştır [32]. Veri tabanından bir örnek aşağıdaki Şekil 4.1’de görülmektedir.



Şekil 4.1: MNIST Veri seti.

4.2.2 FashionMNIST Veri Seti

Zalando araştırmacıları tarafından 2017 yılında LeCun’un ünlü Mnist veri setinden esinlenerek geliştirilen FashionMNIST veri seti 10 sınıflı bir kıyafet veri setidir. Han Xiao’nun belirttiğine göre [16] Mnist veri kümesinin oluşturulan makine öğrenmesi modellerinin sınanmasında çok yetersiz kalmasından dolayı aynı format ve büyüklükte daha zor bir test yöntemi geliştirme arzusuyla ortaya çıkmıştır.

Veri kümesi 28x28 piksel boyutunda siyah beyaz görsellerden oluşmaktadır. 10 adet farklı giysi sınıfından her birinden 7 bin adet olmak üzere 70 bin adet veri içermektedir. Bunlardan 60 bin tanesi eğitim, 10 bin tanesi test için ayrılmıştır.

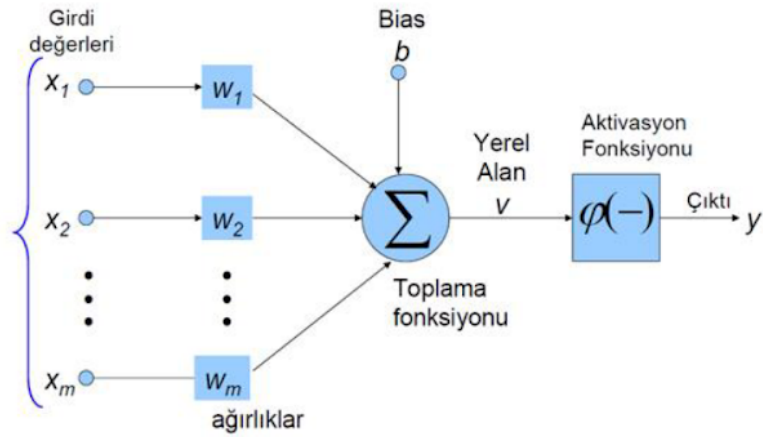


Şekil 4.2: FashionMNIST Veri seti.

4.3 Derin Öğrenme Modelleri

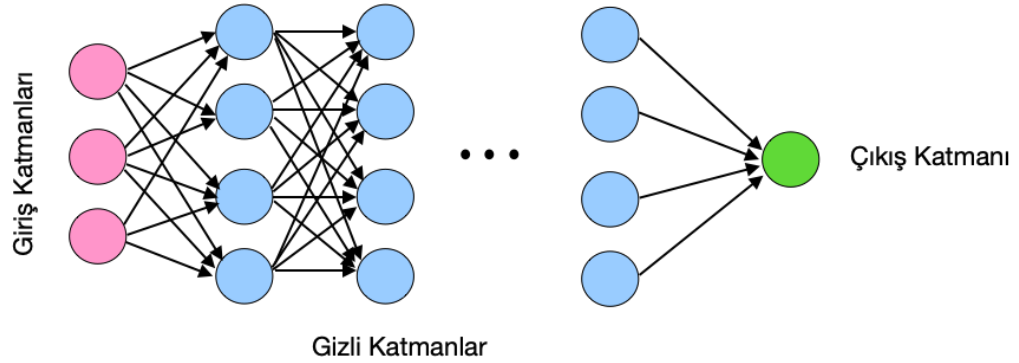
4.3.1 Çok Katmalı Sinir Ağları (MLP)

1943 yılında W. McCulloch ve W. Pitts tarafından matematik ve algoritmalara dayalı bir sinir ağı modeli oluşturulmuştur [33]. Sunulan bu model, beyindeki biyolojik süreçlerin yapay zekâ sinir ağlarına uygulanması fikrine dayanmaktadır [34]. 1958 yılında Rosenblatt [35] tarafından tanıtılan tek hücreli algılayıcı sinir hücreleri ağının temelini oluşturmaktadır. Sinir ağı hücresi ağırlıklarla çarpılan giriş verilerini alarak doğrusal bir kombinasyon oluşturan tek bir çıktı üretir. Çıktı oluşturulurken etkinleştirme fonksiyonu kullanılır.



Şekil 4.3: Sinir ağı hücresi.

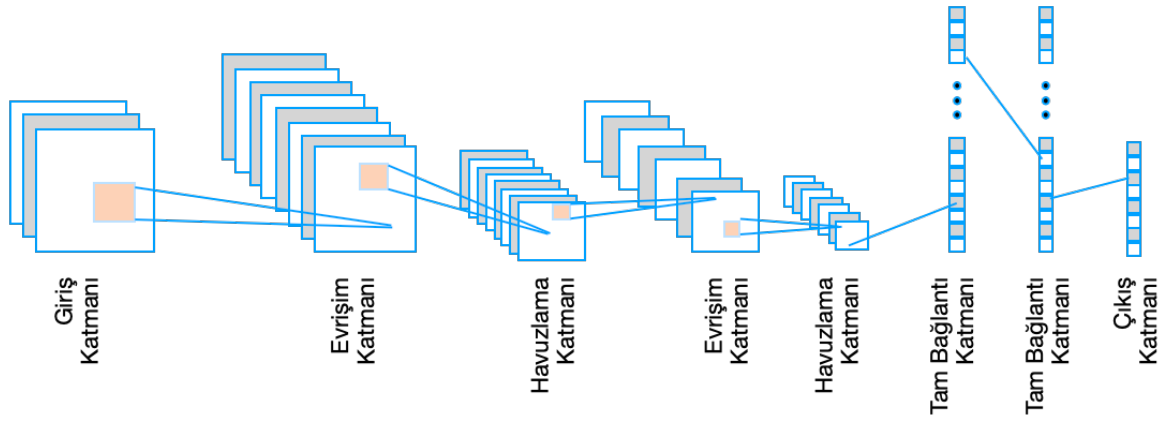
Çok katmanlı algılayıcılar (MLP), giriş ve çıkış arasına yerleştirilmiş gizli katmanlardan oluşan basit sinir ağlarıdır. Tek hücreli algılayıcı sınırlı sayıda eşleştirme gerçekleştirdiği için kullanışlı bir yöntem değildir. MLP girdi, gizli katman/katmanlar ve çıkış katmanından meydana gelir. Girdiden çıkışa doğru yapılan işlemlere ileri yayılım denirken, çıkıştan girdiye doğru ağırlıkların güncellenmesi işlemine geri yayılım denir.



Şekil 4.4: MLP Sinir ağı.

4.3.2 Evrişimsel Sinir Ağı (CNN)

Evrişimsel sinir ağı ilk olarak LeCun ve arkadaşları tarafından, gradyan temelli bir yaklaşım sunulurken ortaya çıkmıştır [36]. Bu yaklaşıma Evrişimsel Sinir Ağı (CNN) adı verilmiştir. Oluşturulan bu yapay sinir ağı modeline ise Lenet adı verilmiştir [37]. Evrişimsel sinir ağı mimarisi, hayvan görsel korteksi ilham alınarak geliştirilmiş en popüler derin öğrenme yaklaşımıdır. CNN'ler iki ya da üç boyutlu diziler üzerinde işlem yapmak için tasarlanmış bir sinir ağı modelidir. Dijital ortamdaki resim ya da video görüntüleri iki ya da üç boyutlu sayı dizilerinden meydana gelen veri biçimleri olduğu için CNN'ler sıklıkla görüntü ve video işleme için kullanılmaktadır. CNN nesne tanıma [38], nesne izleme [39], metin tanıma [40], görüntü sınıflandırma [41], görüntü düzeltme [42], görüntü renklendirme [43] ve çeşitli uygulamalarda yaygın olarak kullanılmaktadır. CNN birbirlerine bağlı çok katmanlı bir algılayıcıdır. CNN ağı katmanları evrişimsel katman, havuzlama katmanı ve tam bağlantı katmanıdır. Evrişimsel sinir ağları çok katmanlı sinir ağlarından farklı olarak giriş matrisleri üzerinde evrişim işlemi gerçekleştirerek öznitelikler çıkarır. Bir sonraki katmana geçmeden önce işlem yükünü azaltmak için bu öznitelikler havuzlama katmanlarında küçültülürler. Son evrişim katmanından tam bağlantılı katmana geçiş yapılırken matris düzleştirme işlemi yapılır. Bir CNN modeli olan Lenet-5 Şekil 4.5'te görülmektedir.



Şekil 4.5: Lenet-5 derin öğrenme modeli.

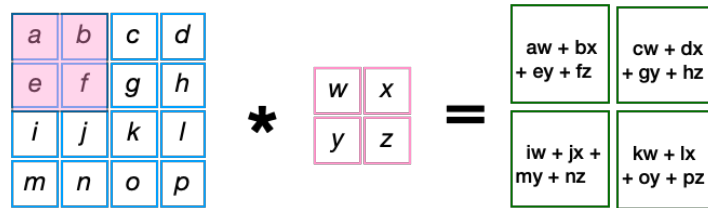
CNN ağlarında her katmanın çıkışı kendinden sonraki gizli katmanın giriş olur. Katmanların derinlikleri katmana ait öznelik sayısını ifade etmektedir.

4.3.2.1 Evrişim Katmanı

CNN ağlarına ismini veren evrişim katmanı bu ağların en temel katmanıdır. Evrişim katmanında bir önceki katmanın ya da girişin özellik haritasını çıkarılır. Görüntü matrisi üzerinde gezdirilen evrişim matrisi (kernel) görüntü matrisi ile evrişim işlemi gerçekleştirilerek özellik haritalarını (feature map) oluşturur. Evrişim işleminden sonra aktivasyon fonksiyonu kullanılarak çıkış elde edilir. İki boyutlu bir matris ve çekirdek matrisinin evrişimi aşağıdaki denklem ile yapılmaktadır [44].

$$s(i, j) = (I * K)(i, j) = \sum_m \sum_n K(m, n) \times I(i + m, j + n) \quad (4.1)$$

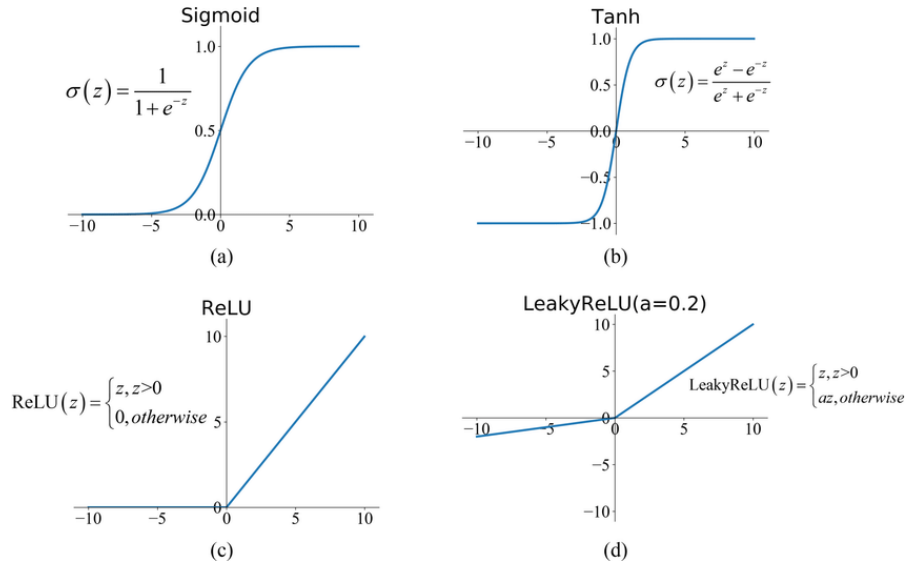
Aşağıdaki görselde 4x4 boyutlarındaki matris üzerinde 2x2 filtre ile evrişim işlemi görülmektedir.



Şekil 4.6: Girdi matrisi ile çekirdek matrisinin evrişim işlemi.

4.3.2.2 Aktivasyon Fonksiyonu

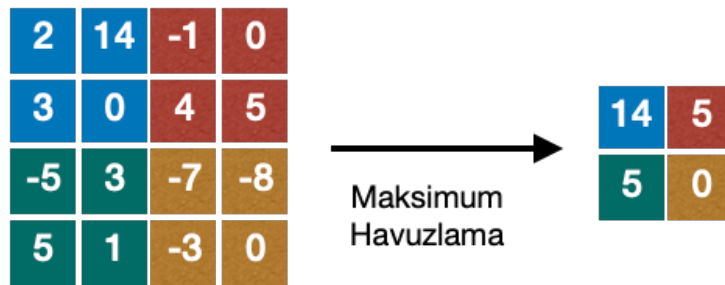
CNN’de evriřim iřleminin sonucu aktivasyon fonksiyonuna gnderilir. Aktivasyon fonksiyonu sinir ađını dođrusal olmaktan ıkararak ğrenme kabiliyetini artırır. CNN zerinde kullanılan aktivasyon fonksiyonlarından bazıları ařađıda grlmektedir.



Őekil 4.7: (a) sigmoid, (b) Tanh, (c) ReLu ve (d) LeakyReLu aktivasyon fonksiyonları.

4.3.2.3 Havuzlama Katmanı

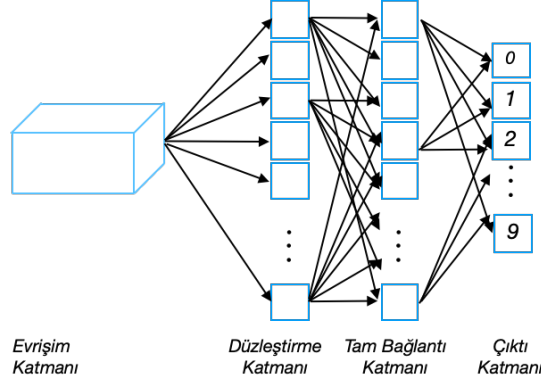
Havuzlama katmanı, evriřim katmanlarından hemen sonra uygulanır ve ađda ilerleyen veri sayısını indirgemeyi amalar. Bu katman, zellik haritasının derinliđini deđiřtirmezken, yksekliđini ve geniřliđini azaltarak veriyi daha kk boyutlara dřrr. Yaygın kullanılan havuzlama yntemlerinden bazıları maksimum (max pooling) ve ortalama havuzlama (average pooling) yntemidir. Maksimum havuzlama ynteminde ekirdek matris iindeki en byk deđer havuzlama katmanının ıkıřına alınır.



Őekil 4.8: Maksimum havuzlama katmanı.

4.3.2.4 Tam Bağlantı Katmanı

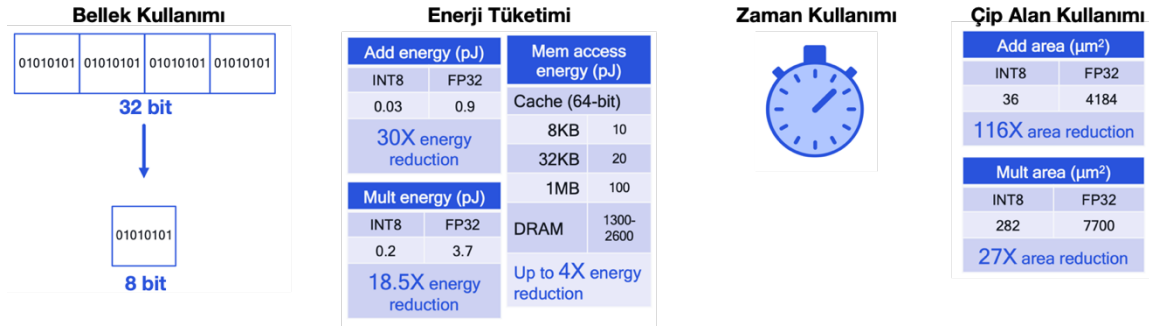
Tam bağlantı katmanı (FC), sinir ağlarında tüm girişlerin diğer nöronlara bağlı olduğu bir yapıdır. CNN ağlarında evrişim ve havuzlama katmanı özellik çıkarımı yaparken tam bağlantı katmanı (FC) sınıf skorlarını optimize ederek sınıflandırma işlemini gerçekleştirir. Girişteki bilginin ağdan geçerek çıkışta istenen sınıfa uygunluk oranını verir. Genel olarak CNN ağlarının sonunda yer alır ve 2 boyutlu özellik haritalarını 1 boyuta dönüştürür.



Şekil 4.9: Tam bağlantılı katman.

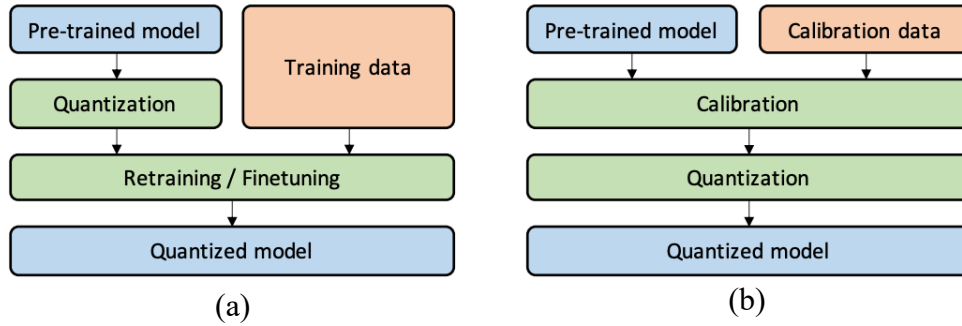
4.3.3 Nicelleştirilmiş Sinir Ağı (QNN)

Niceleme; derin öğrenme modellerinin hesaplama ve bellek yükünü azaltmak için parametre ve verileri düşük bit seviyelerine düşürme işlemidir. Genellikle derin öğrenme algoritmalarında 32 bit kayan nokta veri tipleri kullanılmaktadır. Sinir ağında verileri ve parametreleri düşük bit genişliğinde tam sayılara dönüştüren düşük bit niceleme işlemi ile gerçekleştirilen CNN ağlarına Nicelleştirilmiş Sinir Ağı (QNN) denir. CNN algoritmaları milyonlarca kayan nokta parametresi ve tek bir görüntüyü tanımak için milyarlarca kayan nokta işlemi içerebilir [45]. Örneğin ImageNet 2012 kazanan proje 244 MB'lık parametre ve görüntü başına 1,4 GFLOP işlem yaparken, 2014 yılında kazanan proje ise 552 MB parametre ve görüntü başına 30,8 GFLOP işlem yapmıştır [46]. Derin sinir ağlarının yüksek hesaplama ve bellek kapasitelerine ihtiyaç duyması bu uygulamaların mobil cihazlarda kullanımını ciddi oranda engellemektedir. Niceleme, bir modelin doğruluğunu orijinal modele yakın tutmasına rağmen daha az işlem ve bellek kaynağı tüketmesine olanak tanır [47]. Yapay zekâ modellerinin nicelemesi bellek tüketiminin yanında güç tüketimi, işleme zaman gecikmesi ve çip üzerinde kullanılan alan açısından önemli faydalar sunar. Nagel M. tinnyML projesinde nicelemenin performansa etkilerini aşağıdaki görselde sunmuştur [48].



Şekil 4.10: Niceleme işleminin derin öğrenme üzerindeki avantajları [48].

Nicelleştirilmiş bir derin öğrenme modeli, tensörlerdeki işlemlerin bazılarını veya tamamını kayan nokta değerleri yerine tam sayılarla yürütür. Eğitim sonrası niceleme (PTQ) ve eğitim esnasında niceleme (QAT) olmak üzere iki tip niceleme vardır. PTQ, eğitim sırasında hesaplanan ağırlıklar ve aktivasyon verilerini niceleyerek önceden eğitilmiş modele yerleştirme sürecidir. QAT, eğitimden önce modele niceleme işlemleri eklemek ve modelin eğitimi sırasında nicelemenin gerçekleştirildiği yöntemdir. QAT model eğitilirken ve ağırlıklar hesaplanırken niceleme yaptığı için geri yayılım algoritmasında rol olarak optimizasyonu üst seviyelere çıkarır.



Şekil 4.11: (a) PTQ ve (b) QAT niceleme akış şeması [62].

Nicelleme işlemi yapılırken aşağıdaki denklemler kullanılır.

$$Qnt_{th} = 2^{N-1} \quad (4.2)$$

N nicelenecek bit genişliğini, Qnt_{th} ise niceleme işlemi sonucunda oluşacak maksimum değeri ifade eder.

$$th = \max |A_{i,j}| \quad (4.3)$$

th Niceleme işlemi yapılacak olan A matrisinin mutlak değeri en yüksek olan elemanın değeridir.

$$k = \frac{th}{Qnt_{th}} \quad (4.4)$$

Ölçek (k) matrisin enbüyük elemanının (mutlak değer olarak), maksimum değere bölünmesi ile elde edilir.

$$QntA = \frac{A}{k} \quad (4.5)$$

Niceleme işlemi A matrisinin ölçeğe bölünmesi ile gerçekleşir. Aşağıda örnek bir matris üzerinde 4 bit niceleme işlemi yapılmıştır.

$$A = \begin{bmatrix} -0.235 & 0.205 & -0.654 \\ 0.567 & 0.709 & 0.432 \\ 0.032 & 0.456 & -0.623 \end{bmatrix}, \quad th = 0.709, \quad N = 4, \quad Qnt_{th} = 8$$

$$k = \frac{0.709}{8} = 0.088625$$

$$QntA \cong \begin{bmatrix} -3 & 2 & -7 \\ 6 & 8 & 5 \\ 0 & 5 & -7 \end{bmatrix}$$

Örnek'te görüldüğü gibi giriş matrisi nicelleştirildikten sonra en büyük değeri 8 olurken, en küçük değer ise -7 olmuştur. Bu değerler 4 bit veri genişliğinde yazılabilecek işaretli sayı aralığıdır. 4 bit niceleme sonucunda matrisin alabileceği değer kümesi (-7, -6, -5, -4, -3, -2, -1, 0, 1, 2, 3, 4, 5, 6, 7, 8) olur.

4.3.3.1 Xilinx Brevitas Kütüphanesi ile Nicelleştirme

Brevitas, Xilinx mühendisleri tarafından PyTorch çerçevesi üzerinde nicelleştirilmiş derin öğrenme ağları tasarlamak için geliştirilen bir kitaplıktır. Kullanımı son derece kolay olan bu kitaplık, doğrusallaştırma, evrişim, havuzlama vb. standart PyTorch katmanlarının nicelleştirilmiş muadillerini kullanıma sunar. Brevitas ile bir model tasarlanırken ağırlık,

giriş, çıkış ve aktivasyon katmanları (WIBOL) için nicelleştirme parametrelerini kullanılarak derin öğrenme modeli oluşturulur. Brevitas ile oluşturulan model Pytorch ile oluşturulan modelle göre daha parçalanmış katmanlardan meydana gelmektedir.

Tablo 4.2: Pytorch fonksiyonlarının Brevitas karşılıkları.

PyTorch	Brevitas
nn.Linear	QuantLinear
nn.Identity	QuantIdentity
nn . Conv1d	QuantConv1d
nn . Conv2d	QuantConv2d
nn. ConvTranspose1d	QuantConvTranspose1d
nn. ConvTranspose2d	QuantConvTranspose2d
nn. MaxPool1d	QuantMaxPool1d
nn. MaxPool2d	QuantMaxPool2d
nn. AvgPool2d	QuantAvgPool2d
nn. AdaptiveAvgPool2d	QuantAdaptiveAvgPool2d
nn. Hardtanh	QuantHardTanh
nn. ReLU	QuantRelu
nn . Sigmoid	QuantSigmoid
nn. Tanh	QuantTanh
nn . Dropout	QuantDropout

Brevitas, eğitim esnasında nicelleştirme (QAT) tekniğini kullanır. Brevitasta QuantWibol; ağırlık (w), giriş (i), bayes (b) ve çıkış (o) katmanlarının nicelleştirilmesi anlamına gelir. Niceleme türü QuantType parametresi ile belirlenir. Bu parametre Int, Int8, Binary ve Ternary olarak, ağırlık, aktivasyon, giriş ya da çıkış katmanlarına ayrı ayrı uygulanabilir. Ayrıca Brevitas ile tasarlanıp eğitilen model, Brevitas Onnx Optimizer ile FINN çerçevesine uygun bir şekilde dışa aktarılabilir.

4.3.4 İkileştirilmiş Sinir Ağı (BNN)

Son zamanlarda yapılan çalışmalar CNN uygulamalarındaki kayan noktalı işlemleri kullanmadan evrişim katmanlarında ağırlıklar ve aktivasyon işlemleri için bir ya da iki bit sayı kullanılarak doğru şekilde sınıflama yapılabileceğini göstermiştir [24 , 49]. İkili sinir ağı olarak isimlendirilen BNN'ler küçük hafıza ihtiyaçları, düşük aritmetik hassasiyeti, yüksek hız ve düşük enerji tüketimleri ile ön plana çıkmaktadır. Ayrıca BNN'ler özellikle FPGA'ler üzerinde gerçeklemek için oldukça caziptirler çünkü neredeyse tüm işlemler ikilik

sistemde (binary) gerçekleştirilebilmektedir. Bu nedenle FPGA üzerinde BNN uygulamalarında saniyede tera operasyonlar (TOPS) seviyesine çıkılabilmektedir.

4.3.4.1 İkileştirme Fonksiyonları

BNN katmanları eğitilirken hem ağırlık hem de aktivasyon fonksiyonlarında -1 ya da 1 değerlerini kullanırlar. Corbariaux ve arkadaşları yapmış oldukları çalışmada deterministik ve stokastik olmak üzere iki farklı ikileştirme fonksiyonu kullanmışlardır. Deterministik yaklaşımda işaret fonksiyonu (sign) kullanmışlardır [24].

$$x^b = \text{Sign}(x) = \begin{cases} +1 & \text{if } x \geq 0, \\ -1 & \text{otherwise,} \end{cases} \quad (4.6)$$

x^b ikileştirme yapılmış ağırlık ve aktivasyon değerlerini ifade eder. x gelen görüntü bilgisindeki değerlerdir. Stokastik yaklaşımda ise sigmoid $\sigma(x)$ fonksiyonu kullanılmaktadır.

$$x^b = \begin{cases} +1 & p = \sigma(x), \\ -1 & 1 - p, \end{cases} \quad (4.7)$$

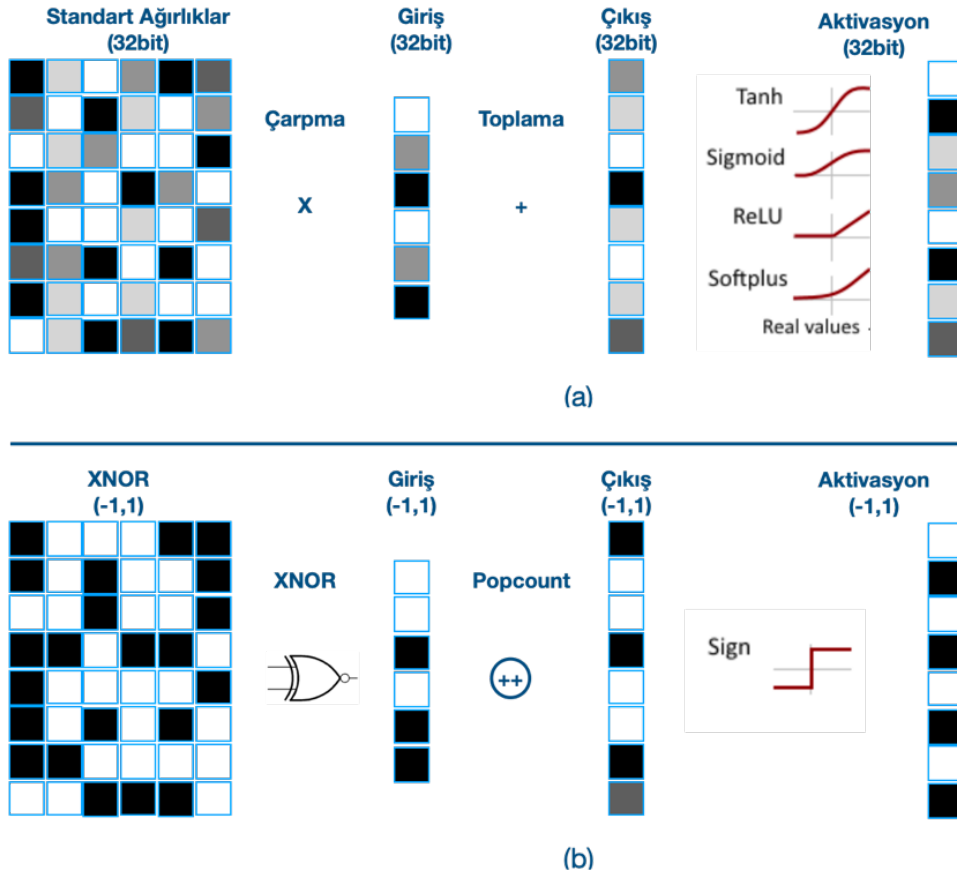
$$\sigma(x) = \text{clip}\left(\frac{x+1}{2}, 0, 1\right) = \max\left(0, \min\left(\frac{x+1}{2}\right)\right) \quad (4.8)$$

Stokastik ikileştirme, deterministik işaret fonksiyonuna göre daha iyi çalışmaktadır, ancak donanım üzerinde çalıştığı sırada rastgele bitler üretmesini gerektirdiğinden uygulanması daha zordur.

4.3.4.2 BNN XNOR İşlevselliği

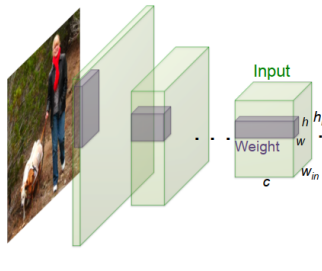
BNN uygulamalarında evrişim işlemi CPU üzerinde gerçekleştirilecek ise hem görüntü hem de çekirdek matris -1, 1 değerlerinde nicelleştirilir ve çarpma işlemi kullanılır. FPGA gibi donanımlar üzerinde yapılacaksa 0, 1 değerlerinde nicelleştirilerek XNOR lojik işlemi uygulanır. İşlemci üzerinde -1 ve -1 çarpımı ile 1 ve 1 çarpımı sonucu 1 verir. Bu durum aynı değerli olan hücrelerin 1 farklıların -1 olarak çıkış vermesini sağlar. XNOR lojik kapıları ise 1, 1 ya da 0, 0 girişlerini 1 olarak çıkartırken farklı girişleri 0 olarak çıkışa aktarır. Böylece evrişim yaparken çarpma işlemi yerine FPGA üzerinde çok daha hızlı uygulanabilen XNOR kapısı kullanılır. Ayrıca CPU üzerinde evrişim yaparken çarpma sonuçları toplama işlemi ile gerçekleştirilirken, FPGA üzerinde bit akışındaki 1 ifadelerini sayan bit sayacı (popcount) donanımı kullanılır. Bit sayacı matris çarpımı ve evrişim işlemlerinde toplama işlemini ortadan kaldırarak hem kaynak kullanımını hem de iş yükünü

azaltır. 32 bit standart ağırlıklarla yapılan bir evrişim işlemi ile XNOR ve bit sayacı kullanılarak yapılan evrişim işleminin farkı aşağıdaki Şekil 4.12’de görülmektedir.



Şekil 4.12: (a) Standart evrişim (b) XNOR evrişim işlemi.

Kim ve Smaragdis yapmış oldukları çalışmada ikilileştirilmiş yani bir ya da sıfır ağırlığa sahip sinir uçlarında XNOR ve bit sayacı işlemleri kullanılarak daha verimli bir BNN çalışması yapmışlardır [49]. Rastegari M. [24] ve arkadaşları XNOR-Net isimli çalışmalarında ikili ağırlıklara sahip gerçek değerli sinir ağırları ile XNOR bağlantılı sinir ağırlarını karşılaştırmışlardır. Gerçek değerli görüntü bilgilerinin ikili ağırlıklarla işleme tabi tutulması ile standart evrişim karşılaştırıldığında kullanılan hafıza boyutunun 32 kat, hesaplama işlemlerinin ise 2 kat azaldığını gözlemlemişlerdir. İkilileştirilmiş görüntü ve ikili ağırlıkları XNOR ve bit sayacı işlemlerine tabi tutarak gerçekleştirdikleri çalışma, standart evrişim ile karşılaştırıldığında hafıza kullanımı 32 kat azalmış, hesaplama işlemleri ise 58 kat azalmıştır. Diğerlerine kıyasla AlexNet modeli üzerinde yapılan bu çalışmada doğruluk oranı %56’dan %44’e düşmüştür [24]. XNOR-NET modeli Şekil 4.13’te görülmektedir.

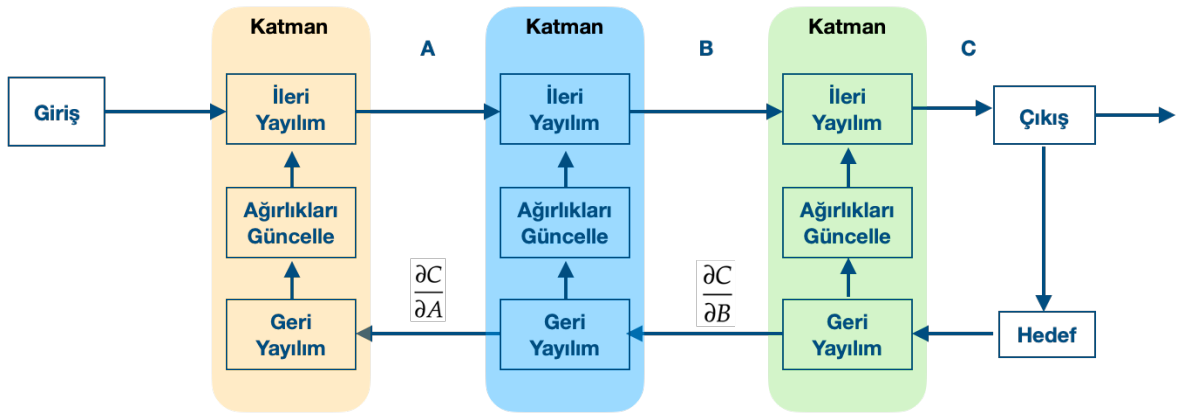


	Network Variations	Operations used in Convolution	Memory Saving (Inference)	Computation Saving (Inference)	Accuracy on ImageNet (AlexNet)	
Standard Convolution	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52	Real-Value Weights 0.12 -1.2 ... 0.41 -0.2 0.5 ... -0.68	+ , - , ×	1x	1x	%56.7
Binary Weight	Real-Value Inputs 0.11 -0.21 ... -0.34 -0.25 0.61 ... 0.52	Binary Weights 1 1 ... 1 -1 1 ... 1	+ , -	~32x	~2x	%56.8
BinaryWeight Binary Input (XNOR-Net)	Binary Inputs 1 -1 ... -1 -1 1 ... 1	Binary Weights 1 1 ... 1 -1 1 ... 1	XNOR , bitcount	~32x	~58x	%44.2

Şekil 4.13: XNOR-Net çalışması kazanç tablosu [24].

4.3.4.3 Gradyanların Yok Olması Problemi ve STE

Yapay sinir ağları eğitilirken ileri yayılma aşamasında düğümlerde ağırlıklar giriş ile çarpılarak aktivasyon fonksiyonuna tabi tutulur. Ardışık olarak ilerleyen bu işlemler çıkışa kadar devam eder. Çıkış ile hedef arasındaki fark alınarak hata bulunur. Geri yayılma aşamasında hatanın aktivasyon fonksiyonuna göre diferansiyeli alınır ve her katman için ağırlıklar güncellenir. İleri ve geri yayılma işlemi blok diyagramda görülmektedir (Şekil 4.14).



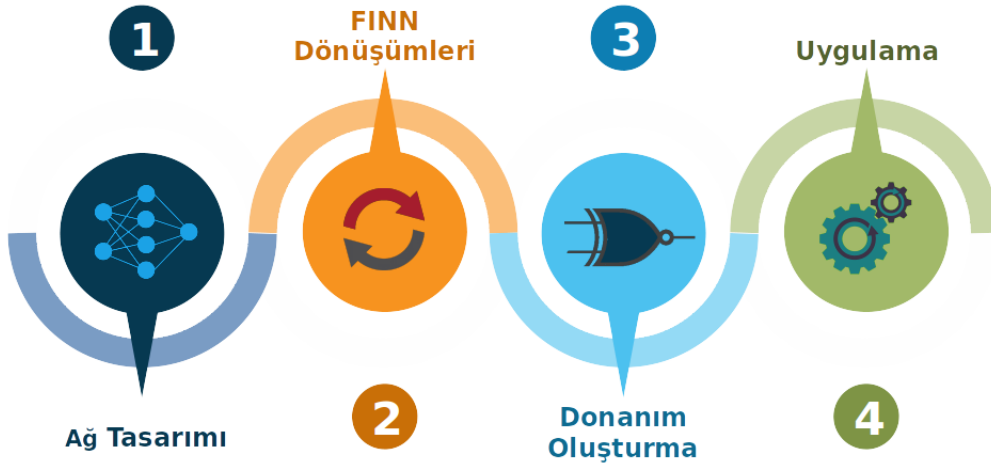
Şekil 4.14: Derin öğrenme ileri ve geri yayılma blok diyagramı.

BNN ağlarında aktivasyon fonksiyonu olarak işaret (Sign) fonksiyonu kullanılmaktadır. İşaret fonksiyonunun türevi hemen hemen her yerde sıfırdır, bu da geri yayılım aşamasında işaret fonksiyonunun kullanımını uyumsuz hale getirir. Böylece ağırlıklar güncellenemez ve gradyanlar yok olur. Bengio, stokastik ayrık nöronlar kullanarak gradyanların tahmini üzerine çalışmalar yapmıştır [50]. Bu çalışmalarda, doğrudan tahmin edicinin (STE) en hızlı

eđitimi gerekleřtirdiđini bulmuřtur. BNN ađlarda ileri yayılma ařamasında iřaret fonksiyonu kullanılırken, geri yayılma ařamasında STE kullanılmıřtır.

4.4 Xilinx FINN erevesi

Xilinx laboratuvarlarında geliřtirilen FINN [28], FPGA'larda leklenebilir ve hızlı nicelenmiř ađları oluřturmaya ynelik bir erevedir. FINN, Pytorch ve Brevitas kullanılarak tasarlanıp eđitilen derin đrenme modelini FPGA zerinde sentezlemek ve uygulamak iin kullanılabilen yeni ve gl bir aratır. FINN, Python ve C programlama dilleri kullanan bir aratır ancak istenildiđinde Vivado HLS ile kod sentezlenebilir. Brevitas ile eđitilmiř nicelenmiř CNN ađlarını iřlemeye ve bunları bir Zynq SoC veya Zynq UltraScale MPSoC zerinde alıřtıracak donanım dosyalarını oluřturmaya olanak tanır. FINN, ardıřık katmanları, AXI Stream arabirimleriyle bir boru hattına bađlanan ayrı hızlandırıcılar olarak uygular. Kullanıcılar, her katman iin uygulanan PE'lerin (İřleme Elemanları) sayısını seebilir. FINN erevesinin kullanımı ařađıdaki blok diyagramda gsterilmiřtir.

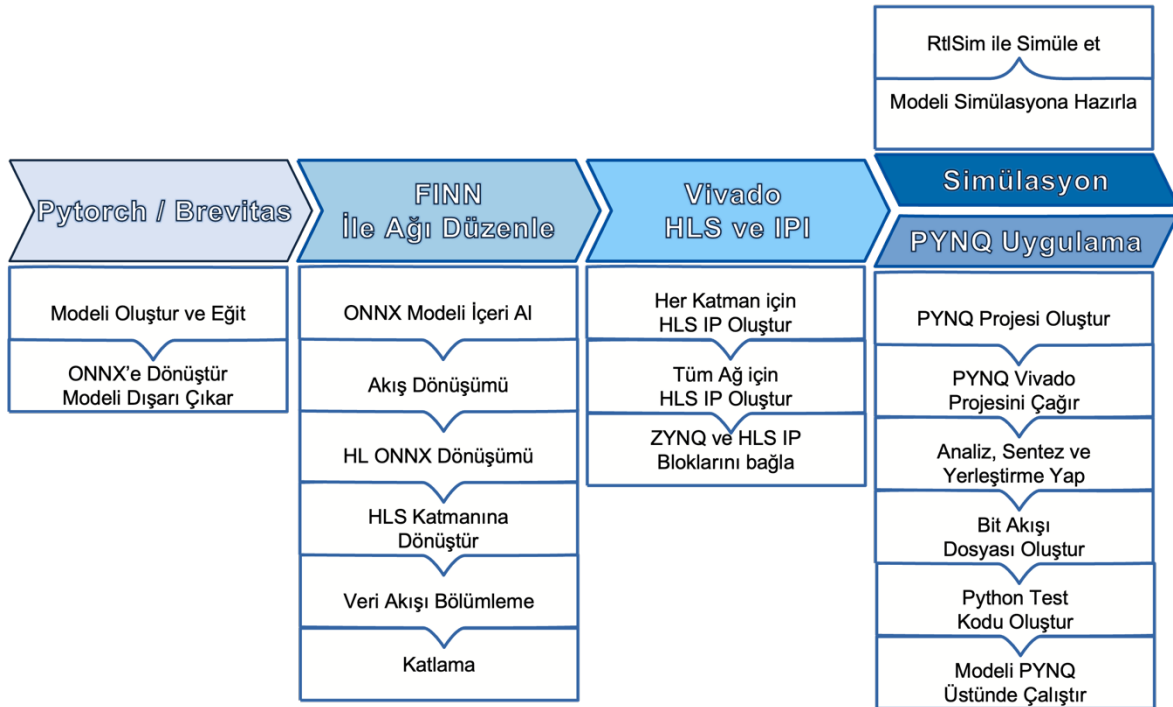


Şekil 4.15: FINN donanım oluřturma akıř řeması.

FINN erevesi ile drt ana grup iřlem blođu kullanılarak bir derin đrenme modeli iin hızlandırıcı tasarlanabilmektedir. İlk ařama Pytorch ve Brevitas kullanarak bir derin đrenme ađı tasarlanır ve eđtilir. Ardından Onnx formatında dıřa aktarılan model ikinci ařama olan FINN dnřmleri ařamasına dahil edilir. Bu ařamada ađ katmanları veri akıřına

uygun hale getirilir. Üçüncü aşamada Vivado kullanılarak FPGA üzerinde IP blokları oluşturulur. Ayrıca bit akışı dosyaları PYNQ'ye aktarılır. Son aşamada ise PYNQ üzerinde ZYNQ işletim sistemi ve FPGA hızlandırıcı birlikte çalıştırılarak ağ hızlandırılır.

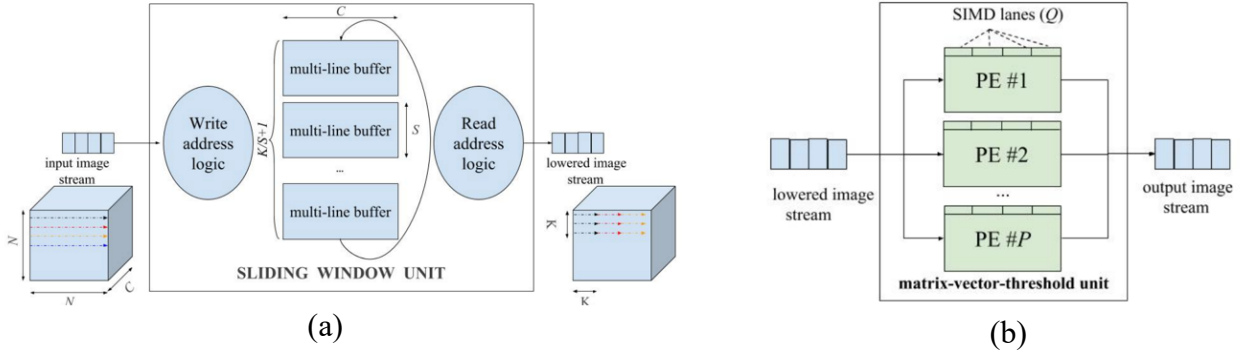
FINN çerçevesinin genel çalışma şekli yukarıdaki gibi özetlense de bu bloklar içinde yapılan ardışıl dönüşüm işlemleri vardır. Tasarlanan Pytorch/Brevitas kütüphaneleri kullanılarak eğitildikten sonra ONNX dönüştürücü ile dışa aktarılır. Ardından ONNX model Xilinx FINN çerçevesinin içine alınır. ONNX model FINN dönüştürücü fonksiyonları peş peşe kullanılarak veri akışı modeline dönüştürülür. FINN çerçevesi kullanılarak hızlandırıcı oluşturmak için yapılan işlemler aşağıdaki akış şemasında görülmektedir. Veri akışı kısmına kadar kullanılan dönüşümler uygulama kısmında dönüşüm grafikleri ile daha detaylı anlatılmıştır. Buradan sonraki en önemli dönüşüm katlama bloğunda gerçekleştirilir. Çünkü bu bölüm ağın kaynak kullanımını ve hızlandırma seviyesini belirleyen paralelleştirme işlemini barındırır.



Şekil 4.16: FINN dönüşüm fonksiyon işlemleri.

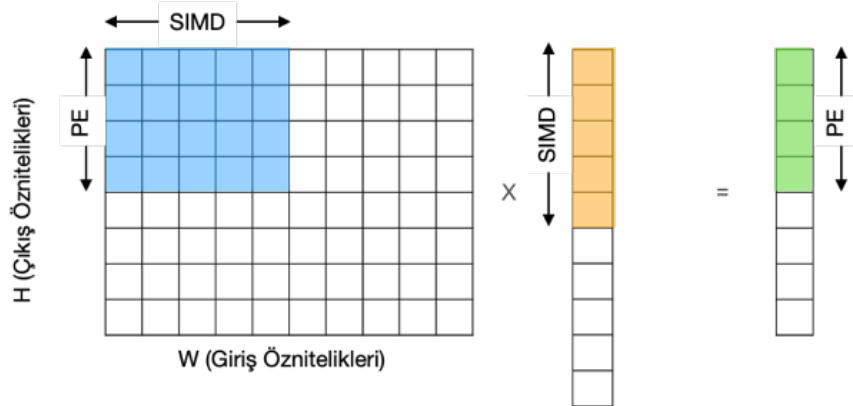
4.4.1 FINN Katlama İşlemi

FINN evrişimsel sinir ağları (CNN) içinde bulunan evrişim işlemini iki temel bileşenle gerçekleştirir. Bunlar matris vektör eşik birimi (MVTU) ve kayan pencere birimidir (SWU) [51]. SWU birimi giriş ve ağırlık matrislerini kayan pencereler haline dönüştürürken MVTU birimi matris çarpma işlemini yapar.



Şekil 4.17: FINN çerçevesi SWU ve MVTU birimleri akış şeması [51].

PE blokları Şekil 4.17'de görüldüğü gibi SIMD sayısı kadar paralel çarpma işlemleri gerçekleştirir. Bu sayı arttıkça işlem daha paralel hale geleceğinden hız artarken kaynak kullanımı da artar. Katlama dönüşümü modelin her katmanı için PE ve SIMD parametresinin belirlenmesi ile gerçekleştirilir. FINN da bir tensör $N \times H \times W \times C$ (Batch sayısı, Yükseklik, Genişlik, Kanal) boyutlarında olur. C kanal sayısı F (katlama) ve P (parallelleştirme) çarpımlarına eşittir. Dolayısıyla katlanmış bir tensörün yeni boyutu $N \times H \times W \times F \times P$ olur. PE çıkış özniteliklerinin, SIMD ise giriş özniteliklerinin tam bölene olmak zorundadır. PE ve SIMD, görüntü ve ağırlık matrislerinin parçalara ayırarak ayrı ayrı çarpma işlemlerini gerçekleştirir. Şekil 4.18'de katlama matris çarpımı 4 ayrı matris çarpımına dönüştürülmüştür.



Şekil 4.18: PE/SIMD ile Matris paralelleştirme.

$NHWC \Rightarrow N = \text{Grup}, \quad H = \text{Yükseklik}, \quad W = \text{Genişlik}, \quad C = \text{Kanal}$

$$C = F \times P \quad (4.9)$$

Kanal sayısı (C), katlama faktörü (F) ve paralelleştirme seviyesi (P) çarpımına eşittir.

$$H \% PE = 0, \quad W \% SIMD = 0 \quad (4.10)$$

PE değeri giriş matrisinin yüksekliğini, SIMD ise genişliğinin tam böleni olmalıdır.

$$F_N = \frac{H}{PE} \quad (\text{Nöron katlama seviyesi}) \quad (4.11)$$

$$F_S = \frac{W}{SIMD} \quad (\text{Sinaps katlama seviyesi}) \quad (4.12)$$

$$T_F = F_N \times F_S \quad (\text{Toplam katlama seviyesi}) \quad (4.13)$$

$$L_{FPS} = \frac{T_F}{F_{clk}} \quad (\text{Saniyede geçen resim sayısı}) \quad (4.14)$$

PE çıkış öznitelik sayısının, SIMD ise giriş öznitelik sayısının tam böleni olmak zorundadır. Hızlandırıcının performansı değerlendirilirken L_{FPS} değeri ile saniyede hızlandırıcıdan geçen görüntü sayısı karşılaştırılır.

5. FPGA TABANLI DERİN ÖĞRENME HIZLANDIRICISI

Bu tez çalışmasında MLP ve Lenet derin öğrenme modelleri için iki farklı metotla üç bölümde FPGA tabanlı hızlandırıcılar geliştirilmiştir. Birinci bölümde Caffe çerçevesi kullanılarak eğitilen MLP ve Lenet modelleri için HLS seviyesinde 8 bit nicelemeli iki hızlandırıcı donanım geliştirilmiştir. Çalışmanın ikinci bölümünde Pytorch ve Brevitas kullanılarak eğitilen MLP ve Lenet modelleri için W1A1, W1A2 ve W2A2 niceleme, yüksek (H) - düşük (L) katlama seviyelerine göre 12 adet hızlandırıcı donanımı tasarlandı. Katman türüne göre katlama seviyelerinin performans ve kaynak tüketimini analiz edebilmek adına tez çalışmasının üçüncü bölümde W1A2 niceleme türünde evrişim ve tam bağlantılı katmanlar için düşük (L), orta (M) ve yüksek (H) katlama kombinasyonları için 9 adet hızlandırıcı donanım geliştirilmiştir. Üç bölümde geliştirilen 23 donanım özellikleri Tablo 5.1’de verilmiştir. Bundan sonraki kısımlarda ayrıntılı olarak açıklanacaktır.

Tablo 5.1: Geliştirilen hızlandırıcı donanımlar özet tablosu.

No	Metot	Çerçeve	Model	Niceleme	Katlama Seviyesi
1	HLS	Caffe	MLP	W8A8	-
2			Lenet-5	W8A8	-
3	FINN	Pytorch/Brevitas	MLP	W1A1	L
4			MLP	W1A1	H
5			MLP	W1A2	L
6			MLP	W1A2	H
7			MLP	W2A2	L
8			MLP	W2A2	H
9			Lenet-5	W1A1	L
10			Lenet-5	W1A1	H
11			Lenet-5	W1A2	L
12			Lenet-5	W1A2	H
13			Lenet-5	W2A2	L
14			Lenet-5	W2A2	H
15	FINN	Pytorch/Brevitas	Lenet-5	W1A2	L-L
16			Lenet-5	W1A2	L-M
17			Lenet-5	W1A2	L-H
18			Lenet-5	W1A2	M-L
19			Lenet-5	W1A2	M-M
20			Lenet-5	W1A2	M-H
21			Lenet-5	W1A2	H-L
22			Lenet-5	W1A2	H-M
23			Lenet-5	W1A2	H-H

5.1 Uygulama 1: HLS Tabanlı 8 bit Sabit Noktalı Hızlandırıcı

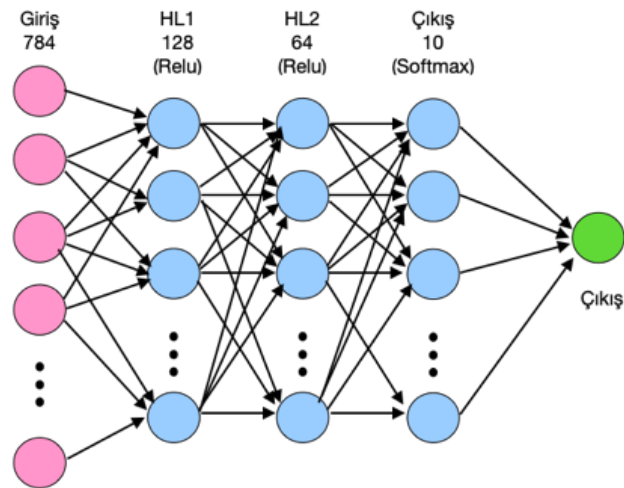
Çalışmamızın birinci bölümünde Caffe çerçevesi MLP ve LeNet mimarileri tasarlandı ve eğitildi. Bu mimariler CPU ve FPGA ortamında, Mnist ve Mnist Fashion veri setleri ile test edildi.

Tablo 5.2: HLS tabanlı 8 bit sabit noktalı FPGA hızlandırıcı test planı.

Model	Veri Seti	Cihaz
MLP	MNIST	Arm Cortex A9
MLP	MNIST	FPGA Zynq 7020
MLP	FashionMNIST	Arm Cortex A9
MLP	FashionMNIST	FPGA Zynq 7020
Lenet	MNIST	Arm Cortex A9
Lenet	MNIST	FPGA Zynq 7020
Lenet	FashionMNIST	Arm Cortex A9
Lenet	FashionMNIST	FPGA Zynq 7020

5.1.1 MLP Mimarisi

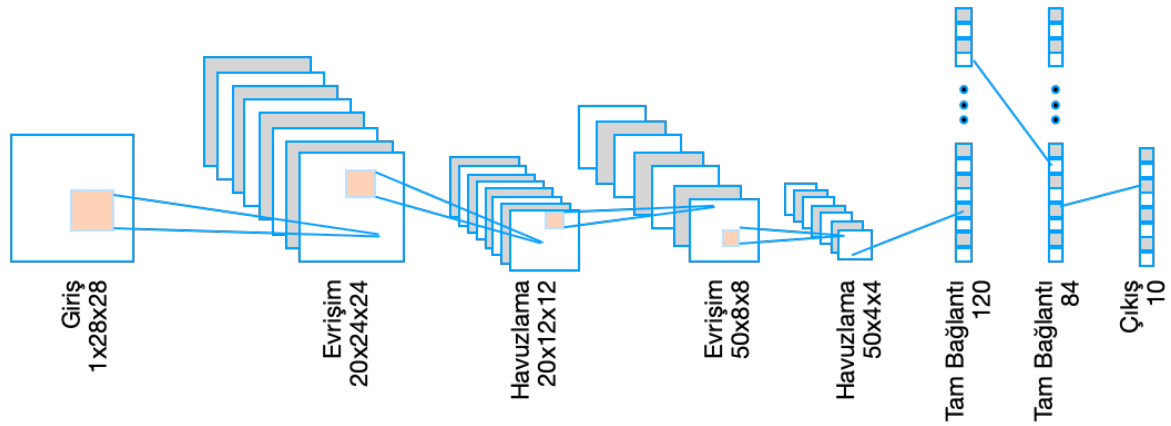
Yapmış olduğumuz çalışmada kullanılan MLP modeli üç gizli katmandan meydana gelmektedir. Giriş katmanı 28x28 giriş görüntü matrisinin düzleştirilmiş halidir. Sırasıyla çıkışları 128, 64 ve 10 olan 3 adet gizli katman sonucunda kayıp fonksiyonu çıkışı belirler.



Şekil 5.1: Çok katmanlı sinir ağı (MLP) modeli.

5.1.2 LENET-5 Mimarisi

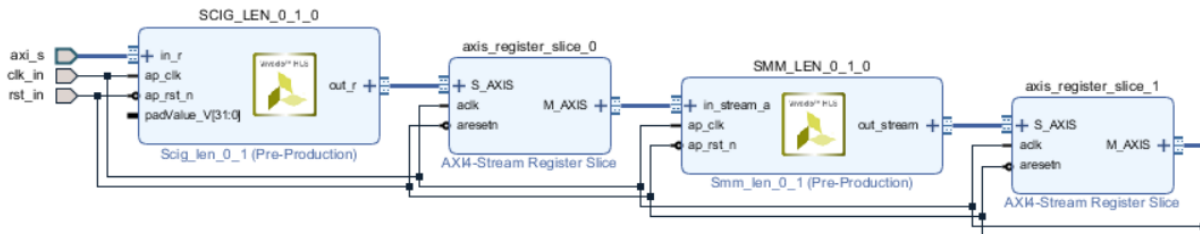
Çalışmamızın ilk bölümünde kullanmış olduğumuz diğer model Lenet-5 modelidir. Bu model çıkışında maksimum havuzlama katmanı bulunan iki adet evrişim katmanı ve 3 adet tam bağlantı katmanından meydana gelir. Giriş görüntüsü 1 kanallı 28x28 boyutlarındadır. Sırasıyla evrişim katmanları 20 ve 50 kanallı, tam bağlantı katmanları ise 120, 84 ve 10 kanalıdır.



Şekil 5.2: Uygulamada kullanılan Lenet-5 derin öğrenme modeli.

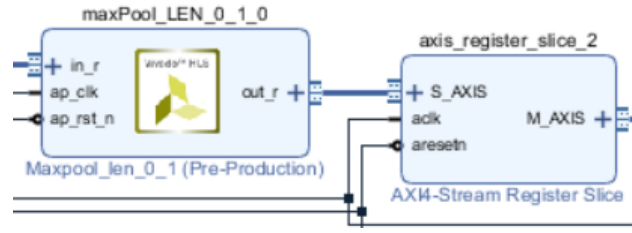
5.1.3 FPGA IP Blokları ve Katmanlar

E. Wang tarafından yapılan çalışmada [52] yüksek seviyeli sentez (HLS) kullanılarak, Xilinx Vivado programı ile evrişim, havuzlama ve tam bağlantı katmanları için IP blokları oluşturulmuştur. Evrişim bloğu iki alt bloktan meydana gelmektedir. SCIG_LEN bloğu girişteki görüntü matrisini im2col fonksiyonu kullanarak matris çarpımları için 3'lü boru hattına dönüştürür. SMM_LEN bloğu ise paralel olarak gelen giriş ve çekirdek matrislerinin çarpma ve toplama işlemlerini gerçekleştirir.



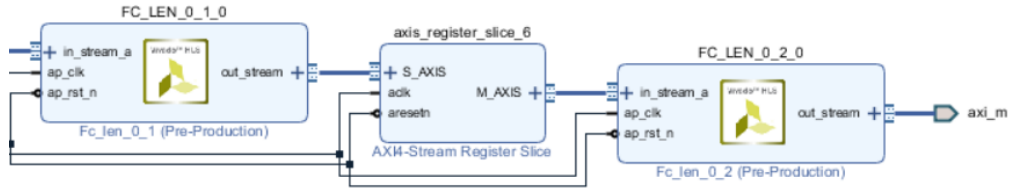
Şekil 5.3: Evrişim katmanı IP bloğu.

Havuzlama katmanı HLS kodları kullanılarak oluşturulur ve AXI ara yüzüne bağlanır. Havuzlama katmanları evrişim katmanlarının sonunda bulunan AXI veri akış bloklarına bağlanır.



Şekil 5.4: Havuzlama katmanı IP bloğu.

Tam bağlantılı katmanlar AXI ara yüz bağlantıları ile Şekil 5.5'te görüldüğü gibi oluşturulurlar.

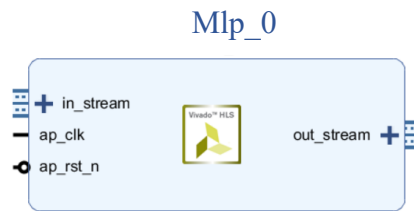


Şekil 5.5: Tam bağlantılı katman IP bloğu.

Çalışmamızın birinci bölümünde E.Wang tarafından tasarlanan bu bloklar kullanılarak MLP ve Lenet modelleri için FPGA hızlandırıcı donanımları tasarlandı ve çalışmamızın ikinci bölümünde FINN çerçevesi ile tasarlanan aynı modellerle karşılaştırıldı.

5.1.4 Uygulama1-MLP Modeli FPGA Bloğu

Bu bloğun girişinde görüntü matrisi tek boyutlu bir diziye dönüştürülür. MLP modelinde art arda üç adet tam bağlantı katmanı kullanılmaktadır. Birincisi öznitelik matrislerini 1x128lik bir diziye dönüştürürken ikinci tam bağlantı katmanı 1x64 boyutunda bir dizi oluşturur ve üçüncü gizli katmana aktarır, üçüncü tam bağlantı katmanı 1x10 boyutunda bir dizi oluşturur. Bu son katman önerilen modelin çıkış katmanıdır.



Şekil 5.6: MLP modeli IP bloğu.

5.1.5 Uygulama1 - Lenet-5 FPGA Bloğu

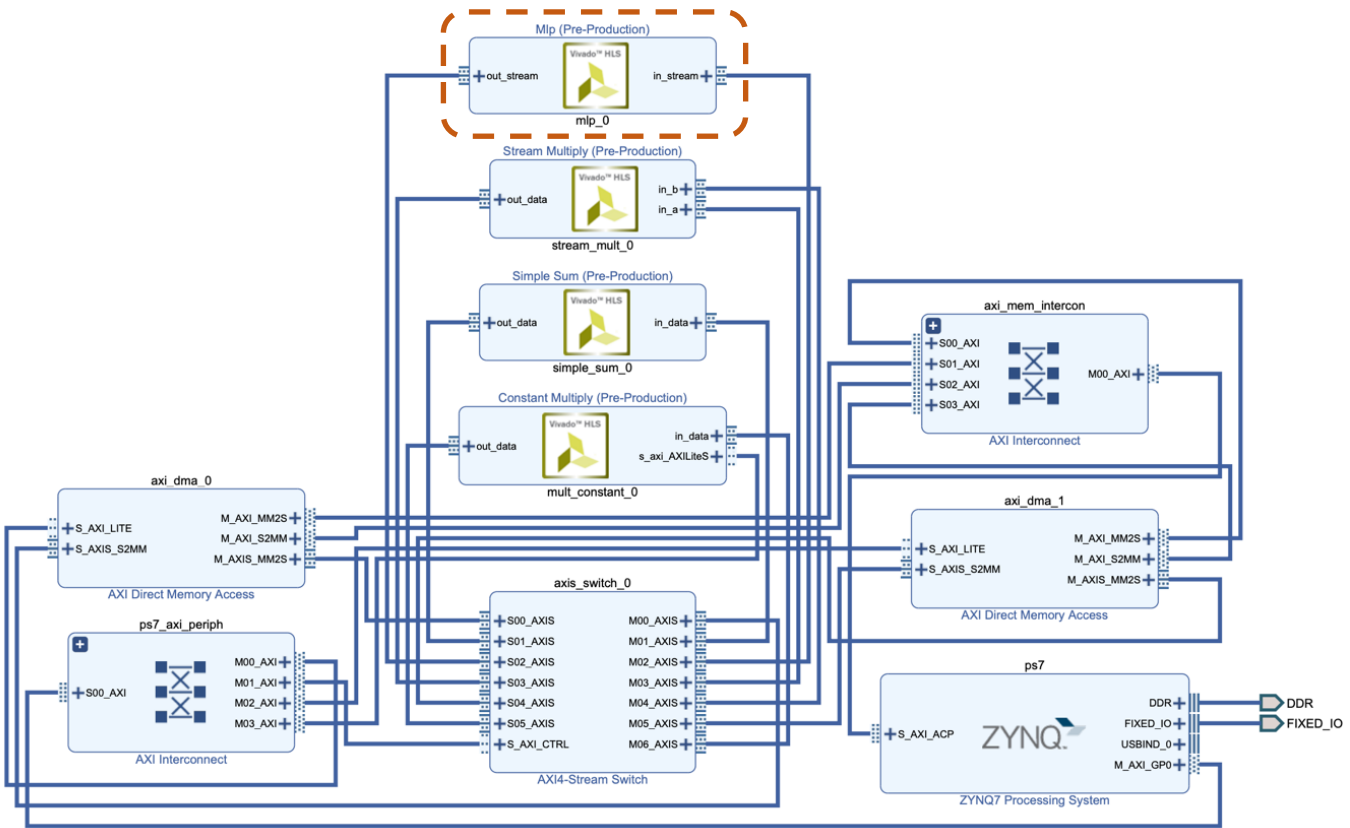
Evrişim ve havuzlama katmanlarında peş peşe geçen görüntü matrisi bu katmanda tek boyutlu bir diziye dönüştürülür. Lenet modelinde art arda üç adet tam bağlantı katmanı kullanılmaktadır. Sırasıyla öznitelik matrislerini önce 1x120 ve 1x84'lük bir diziye dönüştürürken son tam bağlantı katmanı 1x10 boyutunda bir çıkış oluşturur.



Şekil 5.7: Lenet-5 modeli IP bloğu.

5.1.6 MLP ve LENET IP Blokları ZYNQ Bağlantısı

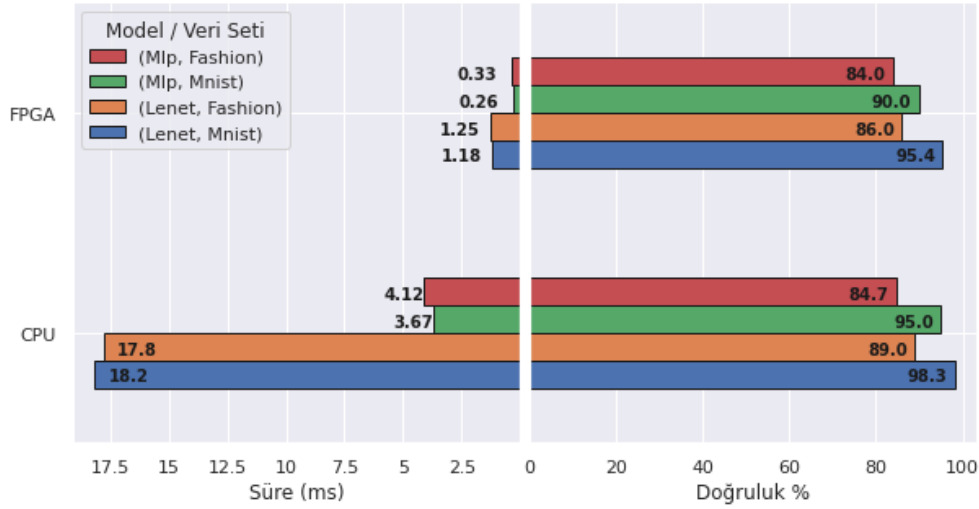
Lenet ve MLP modülleri tek bir IP bloğu olarak sentezlendi ve ZYNQ işletim sistemine AXI ara yüzleri ve diğer donanım ihtiyaçları ile bağlandı.



Şekil 5.8: HLS standart hızlandırıcı donanımı.

5.1.7 Uygulama-1 Bulguları

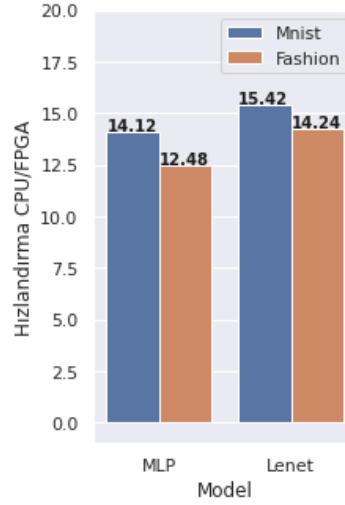
Yapılan FPGA tabanlı hızlandırıcı donanımı iki farklı veri seti ve iki farklı derin öğrenme modeli ile hem CPU hem de FPGA üzerinde test edildi. Elde edilen sonuçlar süre ve doğruluk açısından karşılaştırmalı olarak aşağıdaki grafikte toplandı.



Şekil 5.9: 8 bit MLP ve Lenet modelleri için FPGA ve CPU süre doğruluk grafiği.

Mnist ve FashionMnist veri setleri ile yapılan uygulamalarda doğruluk seviyeleri incelendiğinde MLP ve Lenet modellerinde Mnist %90 üzeri doğruluk verirken FashionMnist veri setinin %80-%90 arasında kaldığı görülmüştür. Bu FashionMnist veri setinin Mnist'e göre öğretilmesi daha zor bir veri seti olduğunu göstermektedir. Lenet modelinde 10000 MNIST görüntüsünün testi CPU üzerinde 18.2 s, MLP modelinde ise 3,6 s sürmüştür. Doğruluk oranı ise MNIST veri seti için MLP modelinde %95, Lenet modelinde %98,3 olarak ölçülmüştür. Lenet modeli %3 oranında doğruluk oranını artırırken MLP modeline göre yaklaşık 5 kat daha yavaş çalışmıştır. Bu süre artışı Lenet modelinde bulunan evrişim işleminin CPU üzerinde oluşturduğu işlem yükünden kaynaklanmaktadır. FPGA üzerinde gerçekleştirilen MNIST veri seti uygulaması incelendiğinde Lenet modelinde 1,18 s, MLP modelinde ise 0,26 s işlem süresi hesaplanmıştır. Doğruluk oranı ise MLP'de %90, Lenet modelinde %95,4 olarak ölçülmüştür. FPGA hızlandırıcı donanımı üzerinde Lenet modeli %5 oranında doğruluğu artırırken yaklaşık 9 kat daha yavaş çalışmıştır. FashionMnist veri seti üzerinde MLP ve Lenet modellerinin doğruluk seviyeleri

incelendiğinde FPGA üzerinde CPU'ya göre yaklaşık %2'lik kayıp görünmektedir. İki veri setinde de yaklaşık sonuçların alındığı gözlemlenmiştir.



Şekil 5.10: MLP ve Lenet modeline göre FPGA/CPU hızlandırma oranları grafiği.

Mnist veri seti ile FPGA üzerinde gerçekleştirilen Lenet modeli hızlandırıcı uygulamasının CPU'dan yaklaşık 16 kat daha hızlı olduğu görülmektedir (Şekil 5.10). MLP modeli üzerinden FPGA ve CPU uygulama süreleri incelendiğinde, FPGA hızlandırıcı donanımı CPU'dan 14 kat daha hızlı çalışmaktadır. FashionMnist veri seti üzerinde FPGA ve CPU karşılaştırmaları yapılıncaya MLP ve Lenet modellerinin hemen hemen aynı sonuçları verdiği görülmüştür. Derin öğrenme işleminin bit akışı modeli ile FPGA üzerinde gerçekleşmesi doğruluk açısından küçük kayıplar vermesine karşın uygulama süresi açısından oldukça hızlıdır.

5.2 Uygulama 2: FINN Tabanlı Hızlandırıcı

Çalışmanın bu kısmında FINN çerçevesi kullanılarak MLP ve Lenet modelleri için farklı ağırlık seviyeleri ve farklı katlama parametreleri ile derin öğrenme hızlandırıcıları tasarlanmıştır. Tasarlanan hızlandırıcılar kaynak kullanımı ve hızlandırma faktörleri açısından analiz edilmiştir. Ayrıca FINN ile geliştirilen hızlandırıcılar önceki uygulamada geliştirilen HLS tabanlı hızlandırıcı ve geçmişte yapılan hızlandırıcı çalışmaları ile karşılaştırılmıştır. Bu bölümde ilk olarak MLP ve Lenet modelleri için FINN çerçevesi kullanılarak her iki model içinde üç farklı ağırlık (W), aktivasyon (A) niceleme seviyesi ve

2 farklı katlama parametresine göre 12 farklı hızlandırıcı donanımı tasarlanmıştır. Bu hızlandırıcı donanımları Mnist ve FashionMnist veri setleri üzerinde uygulanmıştır.

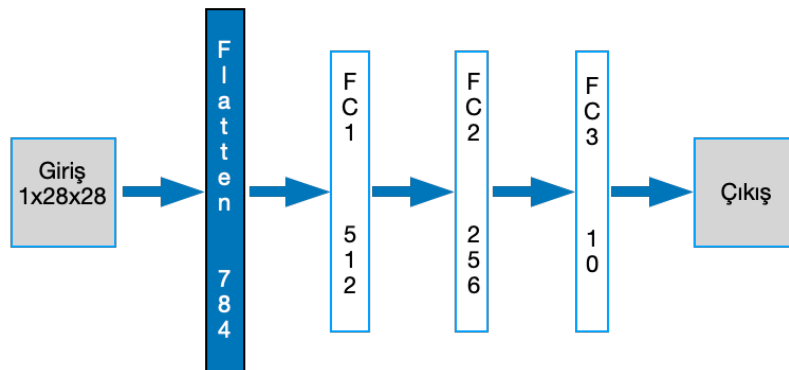
Tablo 5.3: Üç farklı nicelleme seviyesinde MLP ve Lenet modeli için oluşturulan hızlandırıcı test planı.

Model	Nicelleme	Katlama
MLP	W1A1	L
MLP	W1A1	H
Lenet	W1A1	L
Lenet	W1A1	H
MLP	W1A2	L
MLP	W1A2	H
Lenet	W1A2	L
Lenet	W1A2	H
MLP	W2A2	L
MLP	W2A2	H
Lenet	W2A2	L
Lenet	W2A2	H

5.2.1 MLP Hızlandırıcı

5.2.1.1 MLP Model Oluşturma ve Eğitim

Pytorch ve Brevitas kütüphanesi kullanılarak üç tam bağlantı katmanından oluşan MLP derin öğrenme modeli oluşturulmuştur. Modelin ilk gizli katmanı 512, ikinci katman 256 ve çıkış katmanı 10 bağlantı noktasından oluşmaktadır.



Şekil 5.11: MLP hızlandırıcı blok şeması.

Model ağırlık ve aktivasyon bit niceleme sayısına göre niceleme türünü otomatik olarak belirlemektedir. Örneğin ağırlık 1 bit ve aktivasyon 1 bit (W1A1) seçilmişse niceleme türü ikilik (QuantType.BINARY) olurken, niceleme parametresi W2A2 seçilince niceleme türü QuantType.INT olmaktadır. Niceleme türü belirleme için kullanılan algoritmanın Pseudo kodu aşağıdaki tablodaki gibidir.

Tablo 5.4: Niceleme türü belirleme algoritması Pseudo kodu.

```

1: function quant_type ( bit_width ) #bit genişliğine göre niceleme tipi belirleme fonksiyonu
2:   if bit_width = {} then
3:     return QuantType.FP
4:   else if bit_width =1 then
5:     return QuantType.BINARY
6:   else then
7:     return QuantType.INT

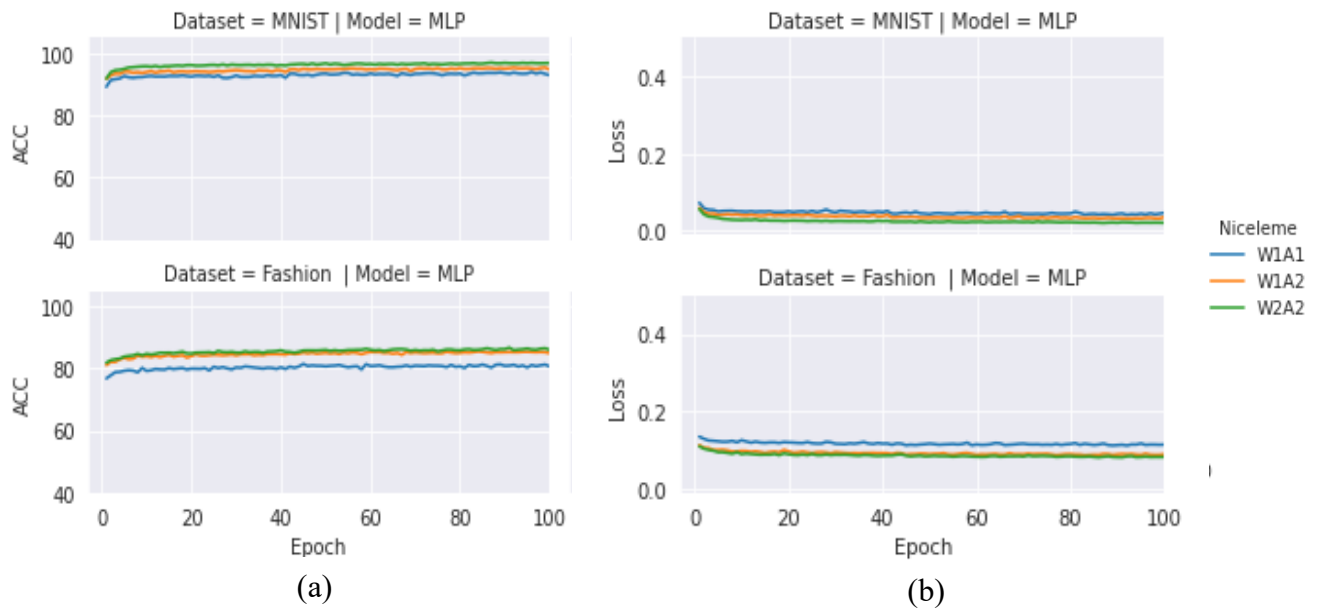
```

Pytorch ile niceleme yapılmadan oluşturulan MLP modeli kodu ile Brevitas kütüphanesi kullanılarak nicelemeli oluşturulan modelin sınıf kodları Tablo 5.5’te verilmiştir.

Tablo 5.5: Pytorch / Brevitas MLP bloğu oluşturma kodu.

Pytorch MLP Kodu	Brevitas MLP Kodu
<pre> class MLP(nn.Module): def __init__(self): super().__init__() self.layers = nn.Sequential(nn.Flatten(), nn.Linear(784, 512), nn.ReLU(), nn.Linear(512, 256), nn.ReLU(), nn.Linear(256, 10)) def forward(self, x): return self.layers(x) </pre>	<pre> class MLP(Module): super(MLP, self).__init__() self.features = ModuleList() self.features.append(QuantIdentity(BINARY,1)) self.features.append(Dropout(p=DROPOUT)) in_features = reduce(mul, in_features) <i>#QuantLinear(inFeatures,outFeatures,weight_bit,QuantType)</i> self.features.append(QuantLinear(784,512,1,BINARY)) self.features.append(QuantLinear(512,256,1,BINARY)) self.features.append(BatchNorm1d(in_features)) self.features.append(QuantIdentity(BINARY,1)) self.features.append(Dropout(p=DROPOUT)) self.features.append(QuantLinear(256,10,1,BINARY)) self.features.append(TensorNorm()) for m in self.modules(): if isinstance(m, QuantLinear): torch.nn.init.uniform_(m.weight.data, -1, 1) def forward(self, x): x = x.view(x.shape[0], -1) x = 2.0 * x - torch.tensor([1.0], device=x.device) for mod in self.features: x = mod(x) return x </pre>

Yukarıdaki kodlarda görüldüğü üzere Brevitas ile nicelenebilir bir model oluşturmak Pytorch çerçevesi ile bir model oluşturmaya göre biraz daha karmaşıktır. Bunun nedeni eğitim esnasında niceleme (QAT) yapabilmek için tüm katmanlarda niceleme parametre ve fonksiyonlarının tanımlanmasının yanı sıra, giriş ve çıkış niceleme işlemlerini de barındırmasıdır. Oluşturulan model test tablosunda belirtilen ağırlık ve aktivasyon niceleme parametreleri ve veri seti çeşitlerine göre ayrı ayrı eğitilmiştir. Bu eğitimler sırasında 100 eğitim tekrarı (epoch) kullanılmıştır. Model eğitim süreci doğruluk verileri her eğitim tekrarı için kaydedilmiş ve aşağıdaki grafikte gözlemlenmiştir.



Şekil 5.12: MLP modeli MNIST ve FashionMNIST eğitim aşaması (a) doğruluk grafiği (b) kayıp grafiği.

Yine aynı şekilde eğitim kayıpları her eğitim tekrarı için kaydedilmiş ve sonuçlar aşağıdaki grafikte görüldüğü gibi olmuştur.

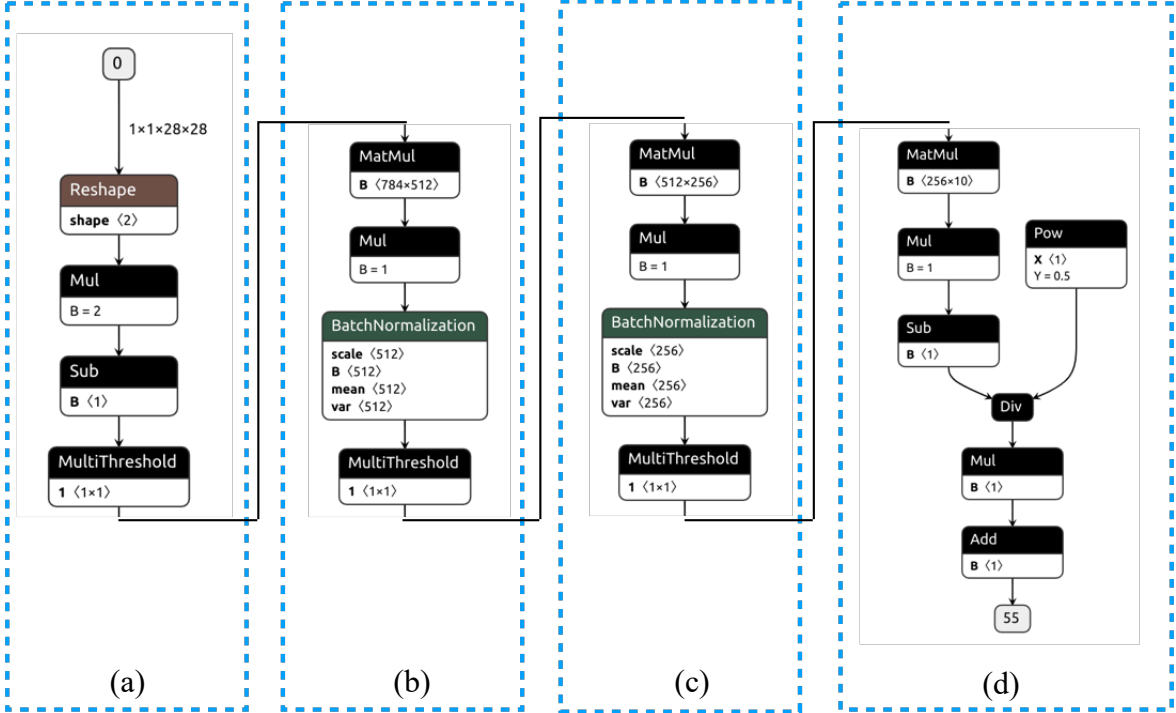
5.2.1.2 MLP Modeli Hızlandırıcı FINN Dönüşümleri

Pytorch ve Brevitas kullanılarak oluşturulan ve eğitilen model FINN çerçevesinde kullanılabilmesi için ONNX biçimine dönüştürülerek, ağırlık ve parametrelerle birlikte dışa aktarılır. Dışa aktarma işlemi aşağıdaki kod bloğu ile best.tar isimli modelin en iyi durumunu tutan kayıttan yüklenerek gerçekleştirilir.

Tablo 5.6: Brevitastan FINN'a ONNX Dönüşüm Kodu.

```
import brevitast.onnx as bo
eniyei_path= "./mlp_w1a1/best.tar"
model = MLP()
eniyei = torch.load(eniyei_path)
model.load_state_dict(eniyei ['state_dict'])
net=model.eval()
path="./finn_model/"
in_shape = (1, 1, 28, 28)
bo.export_finn_onnx(net, in_shape, path)
```

Yukarıdaki kod FINN içine aktarılarak FPGA hızlandırıcı donanımı için gerekli dönüşümler sıralı olarak yapılır. Yüklenen model Netron programı ile görselleştirilmiş ve aşağıdaki şekil elde edilmiştir.



Şekil 5.13: Brevitas ile oluşturulmuş MLP modeli Netron uygulaması üzerinde açılan görünümü.

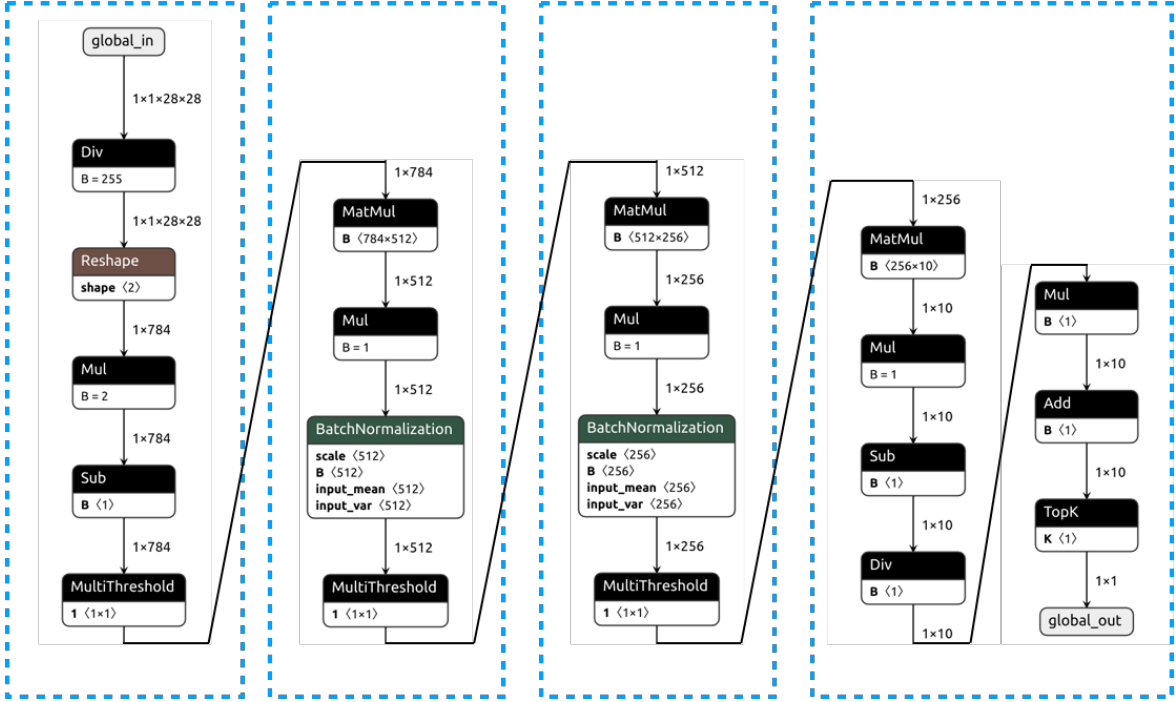
Niceleme yapılmadan 3 katmandan meydana gelen MLP modelinin Brevitas ile nicelemeli olarak oluşturulunca çarpma (Mul), çıkarma (Sub), çoklu eşikleme (MultiTreshold), matris çarpımı (MatMul) ve normalizasyon (BatchNormalizasyon) gibi pek çok bloğa parçalandığı görülmektedir. Reshape bloğundan sonra giriş verileri 2 ile çarpılıp 1 çıkartılarak (0,1) aralığından (-1,1) aralığına dönüştürülür. MatMul, Mul, BatchNormalization ve

MultiTreshold blokları MLP modelindeki ilk tam bağlantılı katmana karşılık gelmektedir. MatMul ve Mul gizli katmanda ağırlık matrisi ile katman girişindeki matrisin çarpımını gerçekleştirirken, BatchNorm ve MultiTreshold ise aktivasyon fonksiyonunun işlevini yerine getirir. C bölümü 512 giriş 256 çıkışlı tam bağlantı katmanını oluşturmaktadır. Son bölüm ise 256 giriş 10 çıkışlı çıkış katmanıdır.

FINN'a aktarılan ONNX modeline ilk olarak aşağıdaki dönüşümler uygulanır.

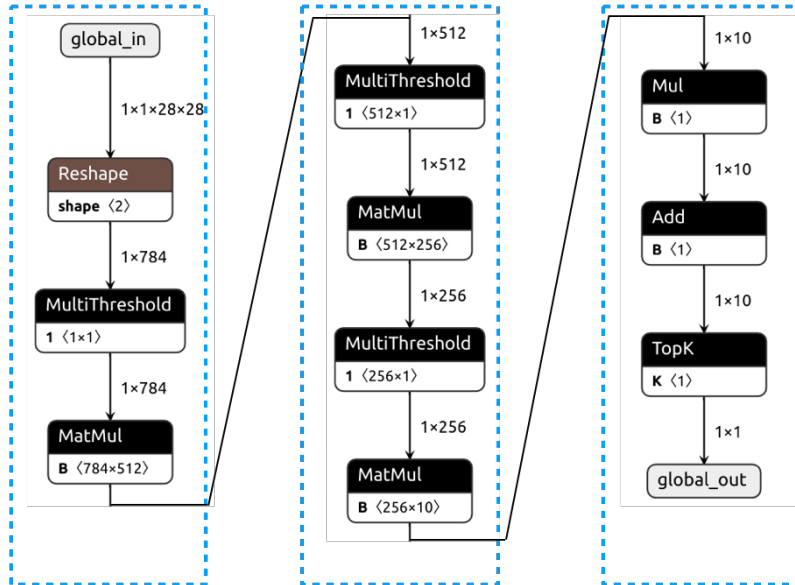
- GiveUniqueNodeNames
- GiveReadableTensorNames
- InferShapes
- InferDataTypes
- FoldConstants
- RemoveStaticGraphInputs

(GiveUniqueNodeNames, GiveReadableTensorNames) dönüşümleri ile grafikteki düğümlere önce benzersiz sayılarla isimler verilir, sonra düğüm adlarına göre okunabilir anlamlı isimler verilir. Ardından (InferShapes, InferDataTypes) dönüşümleri, model özelliklerinden faydalanarak tensörlerin şekillerini ve veri türlerini türetir. Sonraki dönüşüm, düğümlere katlama özellikleri ekleyen FoldConstants dönüşümüdür. RemoveStaticGraphInputs dönüşümü ONNX başlatıcıları için hazırlanmış grafik girişlerini kaldırır. Tüm bu dönüşüm bloğu tidyUp olarak isimlendirilir. TidyUp ardından Pre-Post dönüşüm bloğu uygulanır. Bu blok modele giren 8'er bitlik görüntü verilerini 255'e bölerek 0,1 aralığına normalleştiren bir div bloğu ekler. Ayrıca ağın sonuna en büyük değere sahip çıkışın indeksini hesaplayarak sınıflandırma işlemini gerçekleştiren Top-K bloğu eklenir.



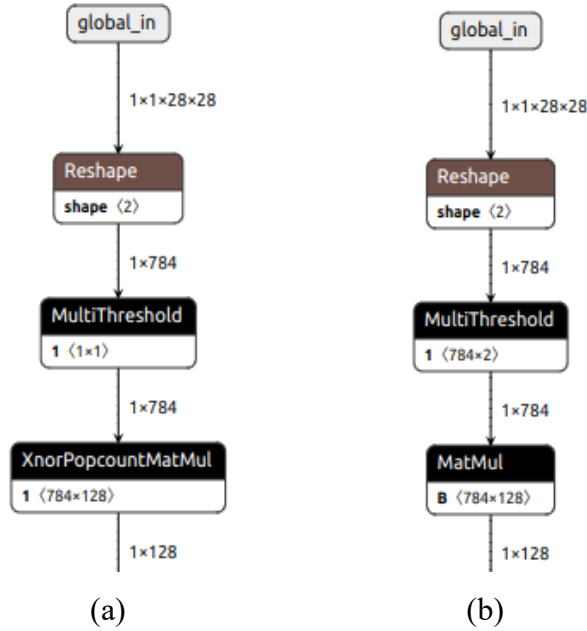
Şekil 5.14: FINN TidyUp ve PrePost dönüşümleri sonucu oluşan MLP modeli.

Sıradaki FINN dönüşümü StreamLine dönüşümleridir. Y. Umurluoğlu ve arkadaşları yapmış oldukları çalışmada [53] StreamLine dönüşümlerinin metodolojisini sunmuşlardır. StreamLine dönüşümünün amacı, kayan nokta işlemlerini tam sayı işlemlerine dönüştürmek ve sonrasında bunları tek bir işlemde daraltarak çoklu eşik düğümlerine dönüştürmektir.



Şekil 5.15: FINN Streamline dönüşümleri sonucu oluşan MLP modeli.

StreamLine dönüşümü ile girişteki Div, Mul ve Sub düğümleri ile matris çarpım düğümlerinden sonra bulunan Mul ve BatchNorm düğümlerinin yok olduğu görülmektedir. HLS katmanlarına hazırlık dönüşümlerinde niceleme parametresi w1a1 gibi nicelemenin 1 bit olduğu durumlarda matris çarpım düğümlerini XnorPopcountMatMul katmanlarına dönüştürür. 1 bit'ten büyük nicelemelerde ise MatMul bloğu kullanılır. Aşağıdaki Şekil 5.16'da sol tarafta W1A1 nicelemeli bir MLP modeli görünürken, sağ tarafta W2A2 nicelemeli model bulunmaktadır. Şekilde de görüldüğü üzere FINN dönüşüm kütüphanesi 1 bit nicelemede XNOR ve Popcount kullanarak ağı daha hızlı ve daha az kaynak tüketebilecek şekilde yapılandırmaktadır.



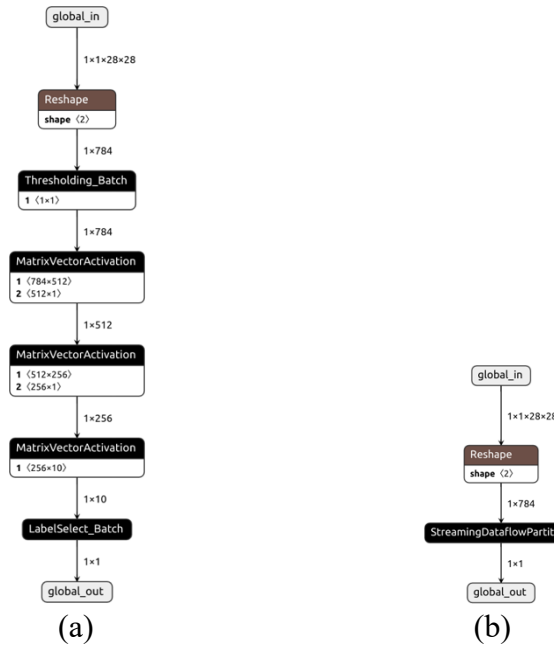
Şekil 5.16: HLS hazırlık dönüşümleri sonucu oluşan modeller.

(a) 1 bit niceleme XNOR / Popcount kullanımı.

(b) diğer niceleme türleri matmul kullanımı.

Sıradaki dönüşüm düğümleri HLS katmanlarına dönüştürür. Bu dönüşüm için finn-hlslib kütüphanesini kullanır. Matris çarpım ya da XnorPopcountMatMul katmanları ile çoklu eşikleme katmanları, MatrixVectorActivation katmanlarına dönüştürür.

CreateDataflowPartition dönüşümü FINN ile bit akışı modeli oluşturmak için kullanılır. Bu dönüşüm sonucu model yalnızca HLS veri akışını temsil eden StreamingDataflowPartition katmanından meydana gelmektedir.



Şekil 5.17: (a) HLS MVU dönüşümü, (b) DataFlow dönüşü sonucu oluşan modeller.

En önemli dönüşümlerden biri de katlama parametrelerinin ağdaki tüm düğümlere eklenmesidir. FINN model sarmalayıcı tüm düğümlerin parametrelerine kolayca erişebilir ve PE / SIMD parametreleri aşağıdaki kodla düğümlere eklenir.

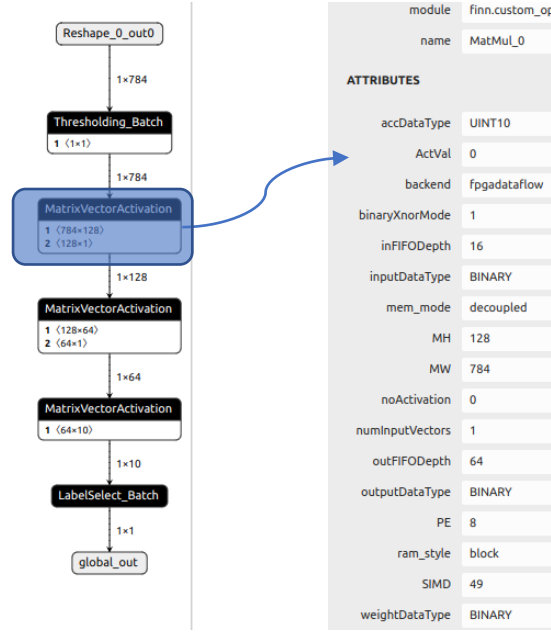
Tablo 5.7: Model'e katlama parametreleri ekleme kodu.

```

fc_layers = model.get_nodes_by_op_type("MatrixVectorActivation")
# (PE, SIMD, in_fifo_depth, out_fifo_depth, ramstyle) for each layer
config = [(16, 49, 16, 64, "block"),
          (8, 8, 64, 64, "auto"),
          (8, 8, 64, 64, "auto"),
          (10, 8, 64, 10, "distributed"),]
for fcl, (pe, simd, ififo, ofifo, ramstyle) in zip(fc_layers, config):
    fcl_inst = getCustomOp(fcl)
    fcl_inst.set_nodeattr("PE", pe)
    fcl_inst.set_nodeattr("SIMD", simd)
    fcl_inst.set_nodeattr("inFIFOdepth", ififo)
    fcl_inst.set_nodeattr("outFIFOdepth", ofifo)
    fcl_inst.set_nodeattr("ram_style", ramstyle)

```

Tablo 5.7'de verilen kod kullanıldıktan sonra model Netron ile incelendiğinde PE ve SIMD parametrelerinin düğümlere eklendiği görülmektedir.



Şekil 5.18: PE/SIMD katlama parametreleri eklenmiş model.

5.2.1.3 MLP Hızlandırıcı Donanımı Oluşturma

FINN çerçevesi ile hızlandırıcı tasarımının üçüncü ana bileşeni FINN donanım oluşturma kısmıdır. FINN hızlandırıcı oluşturmak için FPGA türüne göre iki adet dönüşüm sunmuştur. ZYNQ FPGA'lar için ZynqBuild, Alveo FPGA'lar için ise VitisBuild kullanılır. Bu uygulamada kullanmış olduğumuz PynqZ1 ZYNQ temelli bir kart olduğu için ZynqBuild dönüşümü kullanılmıştır.

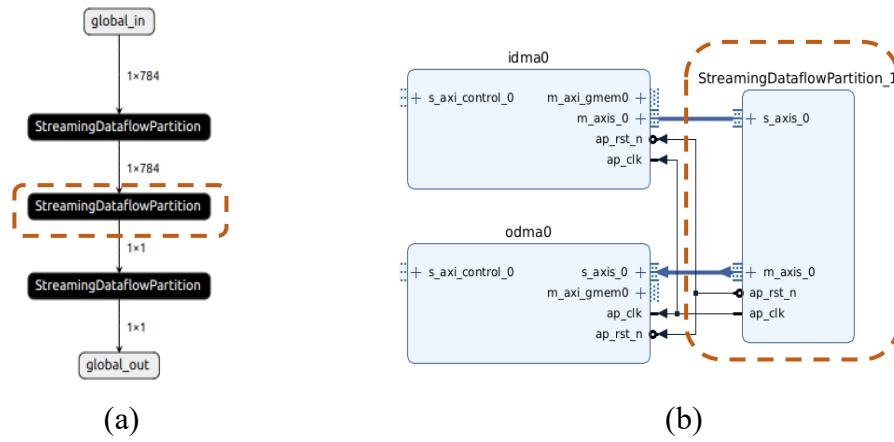
Tablo 5.8: Zynq hızlandırıcı ve Pynq sürücü oluşturma kodu.

```

from finn.transformation.fpgadataflow.make_zynq_proj import ZynqBuild
from finn.transformation.fpgadataflow.make_pynq_driver import MakePYNQDriver
model = ModelWrapper(build_dir+"/mlp_w1_a1_set_folding_factors.onnx")
model = model.transform(ZynqBuild(platform = pynq_board, period_ns = target_clk_ns))
model = model.transform(MakePYNQDriver("zynq-iodma"))
model.save(build_dir + "/mlp_w1_a1_post_synthesis.onnx")

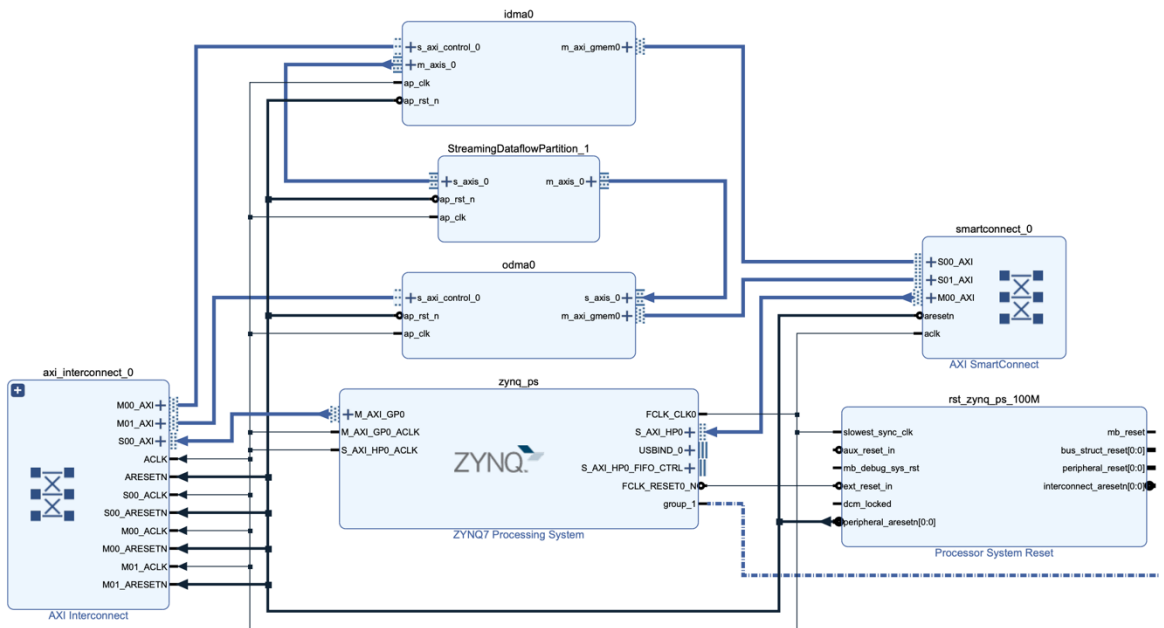
```

Yukarıdaki kod bir Pynq sürücüsü ve FPGA üzerinde modeli gerçekleştiren hızlandırıcı donanımı oluşturulmuştur. Modelin son halinin Netron görüntüsü solda, FPGA üzerinde oluşturulan hızlandırıcı ise sağda görülmektedir.



Şekil 5.19: (a) MLP veri akışı modeli (b) MLP hızlandırıcı IP bloğu.

Şekil 5.19 (a)'da ortadaki bölüm, modelimizdeki tüm katmaları ve bileşenleri içeren bölümdür. İlk ve son bölümler verileri hızlandırıcı ile DRAM arasında taşıyan IDMA ve ODMA bölümleridir. Bu işlem pynq_driver_dir ve vivado_pynq_proj isimli iki adet klasör oluşturur. İlk klasörde bulunan pynq sürücülerini ve hızlandırıcı dosyalarını Pynq platformuna aktarılır. Uygulama sonucunda oluşan hızlandırıcı donanımı ZYNQ işletim sistemi ile AXI ara yüzleri kullanılarak birbirine bağlanır. Böylece PynqZ1 üzerine kurulu Linux tabanlı işletim sistemi görüntü verilerini hızlandırıcı üzerinde paralel olarak derin öğrenme işlemine tabi tutar.



Şekil 5.20: MLP hızlandırıcı ve ZYNQ işletim sistemi bağlantısı.

5.2.1.4 MLP Hızlandırıcı PYNQ Üzerinde Uygulama

Yerel sunucuda Pynq sistemi üzerinde oturum açarak Jupyter Notebook ile uygulama yapılabilmektedir. Ayrıca SSH bağlantısı ile Pynq sistemi üzerinde komutlar işletilerek hızlandırıcı uygulaması uzaktan çalıştırılabilmektedir. Bu çalışmada ikinci yöntem olan SSH bağlantısı kullanılmıştır. SSH bağlantısı için kullanılacak olan IP adresi, kullanıcı adı, şifre ve hedef klasör bilgileri DeployToPYNQ modülü ile modele yazılır. Test için kullanılacak olan veri seti fashion.npz ya da mnist.npz dosyalarından açılarak test değişkenlerine yüklenir. PYNQ dosya sistemi üzerine yazılan validate.py python dosyası SSH ile uzaktan çalıştırılarak hızlandırıcı test edilir. Bu test bize 10000 test görüntüsünün doğruluk oranını ve toplam süreyi verir. Hızlandırıcının verimlilik testi için throughput_test_remote modülü kullanılır. Modül çıktı olarak gönderdiği sözlük Pandas modülü ile tabloya dönüştürülür. Verim testinde toplam süre, saniyede işlenebilecek resim sayısı, DRAM giriş ve çıkış bant genişliği ve hızlandırıcı donanımı içinde belli blokların kullandığı süreler verilmektedir. Test çıktısı Şekil 5.21’de görülmektedir.

Network Metrics	
runtime[ms]	10.508776
throughput[images/s]	951585.634231
DRAM_in_bandwidth[MB/s]	746.043137
DRAM_out_bandwidth[MB/s]	0.951586
fclk[mhz]	100.000000
batch_size	10000.000000
fold_input[ms]	0.179052
pack_input[ms]	0.114441
copy_input_data_to_device[ms]	48.553705
copy_output_data_from_device[ms]	0.605822
unpack_output[ms]	1.107454
unfold_output[ms]	0.205755

Şekil 5.21: Verimlilik testi çıktısı.

MLP hızlandırıcı donanımı üzerinde niceleme seviyelerine ve kullanılan veri setlerine göre aşağıdaki tabloda belirtilen testler gerçekleştirilmiştir.

Tablo 5.9: MLP hızlandırıcı donanımı test planı.

Katlama	Niceleme		
	W1A1	W1A2	W2A2
L	Donanım1	Donanım2	Donanım3
H	Donanım4	Donanım5	Donanım6

5.2.1.5 MLP Hızlandırıcı Bulgular

Uygulamanın bu aşamasında PE ve SIMD parametreleri MLP modeli için aşağıdaki tabloda belirtildiği gibi toplam katlama yüksek (H) ve düşük (L) olacak şekilde iki farklı katlama türü belirlendi. W1A1, W1A2 ve W2A2 niceleme türleri için L seviyesinde katlama yapılmamıştır. H katlama seviyesi ise PYNQ Z1 kartının FPGA kaynaklarını maksimum kullanacak şekilde W1A1 niceleme türü için 64, W1A2 için 128 ve W2A2 için 256 olarak belirlenmiştir. Bu oranlar yapılan testlerle belirlenmiştir.

Tablo 5.10: MLP model katlama parametreleri.

Katlama Grubu	H			L
	W1A1	W1A2	W2A2	Tümü
Niceleme	W1A1	W1A2	W2A2	Tümü
PE	64-32-5	64-32-5	32-16-5	1-1-1
SIMD	98-64-8	49-32-4	49-32-2	1-1-1
Toplam Katlama	64	128	256	-

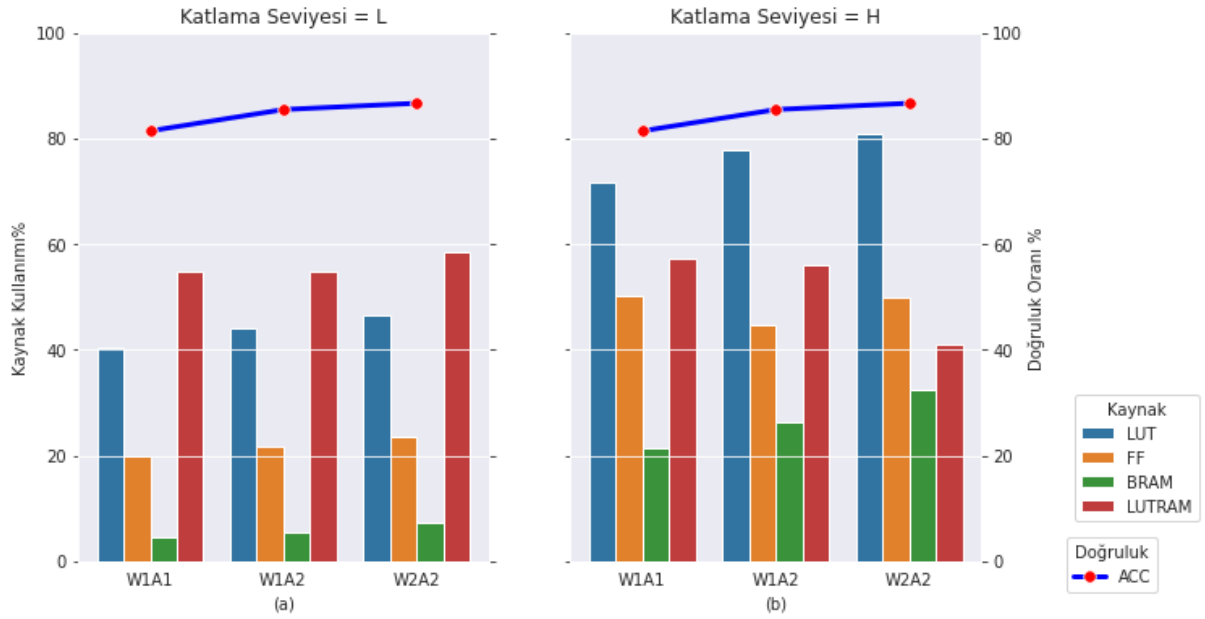
İki farklı katlama parametresi ve üç niceleme çeşidine göre FashionMNIST veri seti ile eğitilen modeller için altı farklı hızlandırıcı donanımının FPGA kaynak kullanımı ve 10000 görüntünün sistemden geçiş süreleri aşağıdaki tabloda verilmiştir.

Tablo 5.11: MLP hızlandırıcı niceleme / katlama oranlarına göre kaynak kullanımı ve verim tablosu.

Model	W1A1		W1A2		W2A2	
	L	H	L	H	L	H
Katlama Konf.	L	H	L	H	L	H
LUT	40,30%	71,71%	44,21%	77,88%	25,67%	70,91%
LUTRAM	54,78%	57,19%	54,82%	55,99%	7,36%	40,87%
FF	19,95%	50,16%	21,72%	44,72%	15,82%	50,02%
BRAM	4,64%	21,43%	5,36%	11,43%	26,79%	32,50%
Süre (ms)	47945	10,5	47789	13,84	47789	27,04
Geçiş (Resim/sn)	208	951585	209	722060	209	369789

Sonuçlar incelendiğinde tam bağlantı katmanlarında katlama faktörünün artması kaynak kullanımını çok fazla artırmamasına rağmen 10000 görüntünün testinin yaklaşık 48000 ms den 10 ms seviyelerine düştüğü görülmektedir. Katlama oranı 64 iken saniyede test edilen resim sayısının yaklaşık 208'den 950000 seviyelerine çıkmaktadır. Bu da sistemin 475x hızlandırılabilirdiği anlamına gelmektedir.

FPGA üzerinde mantıksal işlemler LUT ve FF kaynaklarını kullanırken, bellek işlemleri BRAM ve LUTRAM kaynaklarını kullanır.



Şekil 5.22: MLP hızlandırıcı niceleme ve katlama seviyelerine göre kaynak kullanımı ve doğruluk grafiği.

MLP modeli için H ve L katlama seviyeleri için üç farklı niceleme seviyesinde tasarlanan altı hızlandırıcı donanımının FashionMNIST veri seti ile yapılan testler sonucunda yukarıdaki Şekil 5.22’te görselleştirilmiştir. Dört farklı kaynağın kullanımı bar grafiklerle, üç farklı katlama seviyesi için hızlandırıcının doğruluk seviyeleri çizgi grafiklerle gösterilmiştir. İki katlama seviyesinde de doğruluk seviye grafiği aynı çıkmıştır. Bu sonuç katlama seviyesinin derin öğrenme hızlandırıcısının doğruluk seviyesini etkilemediğini göstermektedir. Bunun aksine hızlandırıcının doğruluk oranının niceleme bit sayısı ile doğru orantılı olarak değiştiği görülmektedir. W1A1 niceleme türünde doğruluk %82 seviyelerinde iken W2A2 niceleme türünde %86 seviyelerine yükselmiştir. L katlama seviyesindeki hızlandırıcılar incelendiğinde niceleme seviyesi arttıkça BRAM ve LUTRAM bellek kaynak kullanımı artarken LUT ve FF gibi mantık kaynaklarının hemen hemen aynı kaldığı görülmektedir. Katlama seviyesi H olan hızlandırıcılarda ise hem bellek hem de mantık kaynaklarının niceleme bit sayıları ile doğru orantılı olarak artmaktadır. H katlama seviyeli hızlandırıcı modellerinden W1A1 ile W1A2 modellerinde BRAM kullanımı aynı olmasına karşın W2A2 nicelemeli hızlandırıcıda BRAM kullanımı yaklaşık iki katına çıkmaktadır. Bu durum bellek kullanımını aktivasyon nicelemesinden çok ağırlıkların nicelemesinin etkilediğini gösterir. W1A1 nicelemede FF ve LUTRAM kullanımının W1A2 nicelemeli

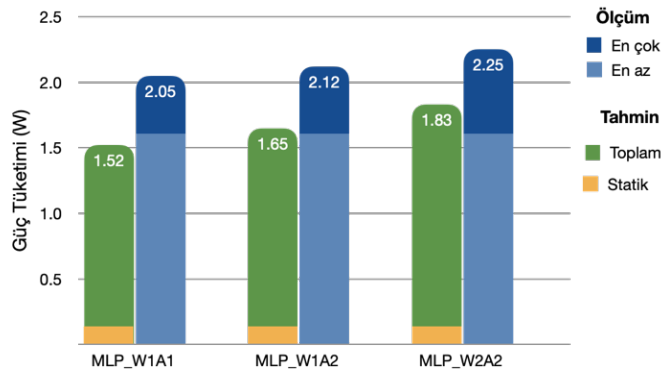
hızlandırıcıdan fazla olmasının sebebi kaynak kullanımının otomatik olarak ayarlanmasından kaynaklanmaktadır.

MLP modeli ve üç farklı nicelme seviyesinde geliştirilen hızlandırıcıların güç tüketimi Şekil 5.23'te görülen USB güç ölçer donanımı kullanılarak boшта ve işlem esnasında fiziksel olarak ölçülmüştür.



Şekil 5.23: Fiziksel ölçümlerde kullanılan USB güç ölçer.

Ayrıca tasarlanan hızlandırıcıların Xilinx Vivado yazılımında bulunun güç tahmini uygulaması kullanılarak güç analizi yapılmış ve USB güç ölçerle yapılan ölçümlerle karşılaştırılmıştır.



Şekil 5.24: Güç tüketimi ölçüm ve Vivado tahmin sonuçları grafiği.

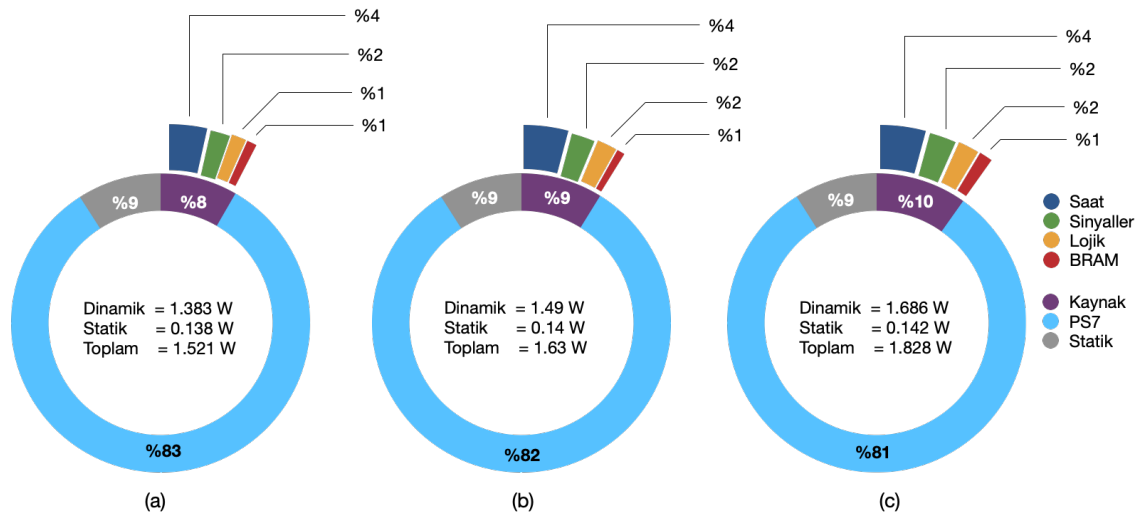
Grafikte hızlandırıcı çalışırken fiziksel olarak ölçülen en az ve en çok güç tüketim değerleri ile Vivado tarafından tahmin edilen statik ve toplam güç tüketimleri görülmektedir. Fiziki ölçümlerin Vivado tahminlerinden 0.5 W yüksek olmasının sebebi Vivado'nun sadece FPGA ve ZYNQ işletim sisteminin tükettiği enerjiyi hesaplamasıdır. Oysaki kullanılan

PYNQ Z1 kartında FPGA dışında genel amaçlı giriş çıkış donanımları bulunmaktadır. USB güç ölçer FPGA ile birlikte bu donanımların tükettiği gücü de ölçmektedir. Güç tüketimleri kaynak türü ve tüketim yerine göre Vivado güç tahmin uygulaması ile hesaplanmış ve Tablo 5.12’de Watt birimi olarak verilmiştir.

Tablo 5.12: MLP hızlandırıcıların Vivado programı güç tüketim tahminleri tablosu.

Tüketim Türü	MLPW1A1	MLPW1A2	MLPW2A2	Tüketim Yeri
Dinamik	0,058	0,093	0,128	Saat
	0,030	0,058	0,108	Sinyaller
	0,024	0,057	0,122	Lojik
	0,015	0,026	0,069	BRAM
	0,127	0,234	0,430	Toplam Kaynak
	1,256	1,256	1,256	PS7
Statik	0,138	0,140	0,142	
Toplam	1,521	1,630	1,828	

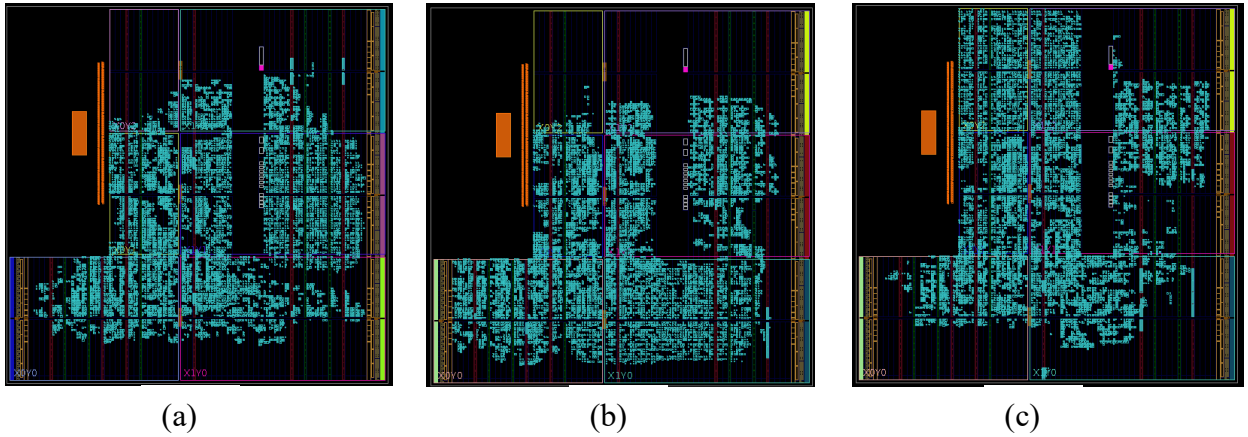
Güç tüketimleri tüketim türüne göre dinamik ve statik olarak iki grupta toplanmıştır. Dinamik tüketimler ise kaynaklar ve PS7 olarak iki gruba ayrılır. Kaynak; saat, sinyaller, lojik elamanlar ve hafıza birimlerini, PS7 ise ZYNQ işletim sistemini temsil eder. Tablodaki tüketimlerin her bir hızlandırıcıda yüzdelik olarak karşılığı Şekil 5.25’te görselleştirilmiştir.



Şekil 5.25: (a) MLP W1A1 hızlandırıcısı (b)MLP W1A2 hızlandırıcısı (c) MLP W2A2 hızlandırıcısı güç tüketimi dağılımı Vivado tahminleri.

Yukarıdaki üç grafik incelendiğinde niceleme seviyesinin artmasının toplam güç tüketimini artırdığı görülmektedir. Güç tüketimleri statik ve dinamik olarak gruplandırılmıştır. Statik tüketim üç hızlandırıcıda da 0.138 W sabit kalmıştır. Dinamik güç tüketimlerinde toplam güç tüketiminin yaklaşık %82'lik kısmını Pynq Z1 kartını kontrol eden PS7 işletim sisteminin tükettiği görülmektedir. Hızlandırıcı donanımının güç tüketimi üstündeki etkisi grafiklerde iç halkada bulunan mor bölgede belirtilmektedir. Hızlandırıcı donanımının kullanmış olduğu saat sinyallerinin, diğer veri haberleşmesi sinyallerinin, lojik donanımların ve hafıza birimlerinin tükettiği enerji dış halkada dört grupta görülmektedir. MLP W1A1 hızlandırıcıda toplam kaynak tüketimi %8'lerde iken W2A2 modelinde bu oran %10 seviyesine çıkmıştır.

Hızlandırıcı tasarımlarında değerlendirilen bir diğer kriter ise alan kullanımıdır. Üç farklı niceleme seviyesine göre tasarlanan MLP hızlandırıcı donanımlarının Vivado uygulamasından alınan alan kullanımları Şekil 5.26'da verilmiştir.



Şekil 5.26: (a) MLP 1W1A (b) MLP 1W2A (c) MLP2W2A Hızlandırıcıları FPGA alan kullanımları.

Yukarıda alan kullanımları verilen üç MLP hızlandırıcıda niceleme seviyesinin artışının LUT, FF ve BRAM gibi kaynakların tüketimini artırmasından dolayı FPGA üzerinde kullanılan alanında doğrusal olarak arttığı görülmektedir.

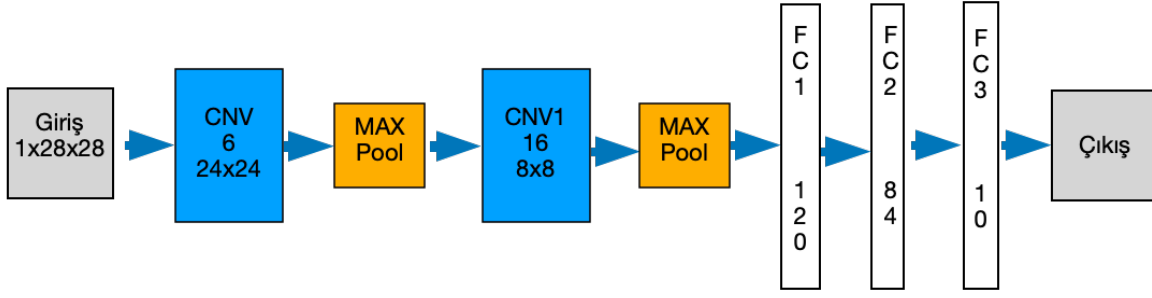
5.2.2 Lenet Hızlandırıcı

5.2.2.1 Lenet Modeli Brevitas ile Oluşturma

Lenet modeli Pytorch / Brevitas kütüphanesi kullanılarak havuzlama katmanına sahip iki evrişim katmanı ve ardından 3 tam bağlantı katmanından oluşmaktadır. Modelin ilk evrişim katmanı 6, ikincisi ise 16 öznelik katmanı çıkarmaktadır. Havuzlama katmanları maksimum havuzlama metodunu kullanır. Temelde tasarlanan model Şekil 5.27'deki gibi olsa da Brevitas bu modeli daha çok parçalara ayırarak oluşturur. MLP modelinde olduğu gibi niceleme tipi ağırlık ve aktivasyon bit sayısına göre kod içindeki fonksiyon tarafından atanır. Modelin Pytorch ve Brevitas kullanılarak oluşturulması aşağıdaki tabloda görülmektedir. Pytorch kullanılarak oluşturulan model daha sade ve sadece katmanlardan oluşurken, Brevitas ile oluşturulan model pek çok bileşene ayrılmıştır.

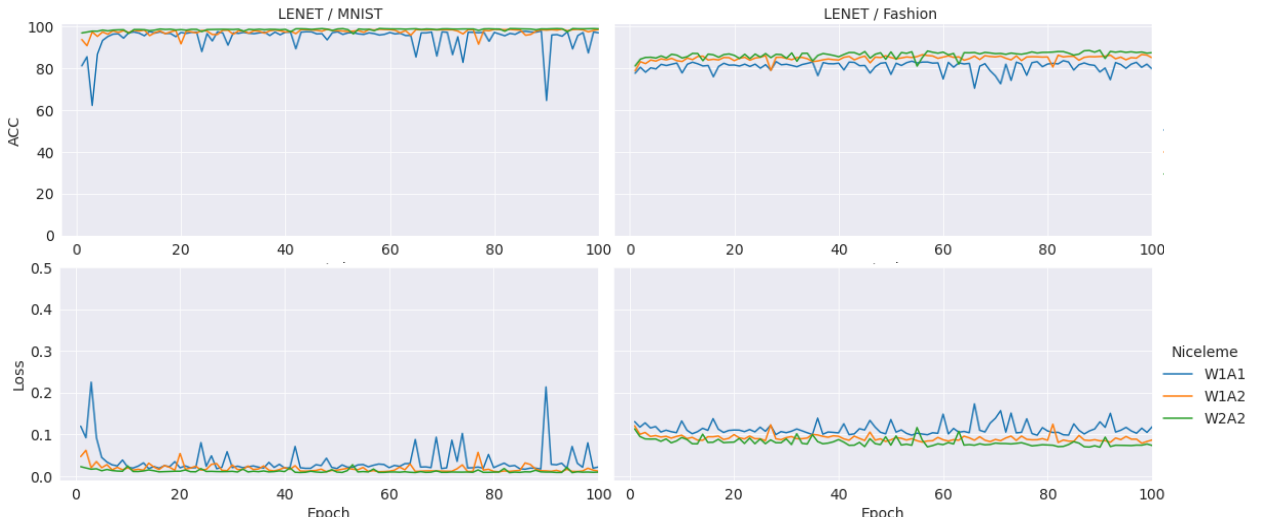
Tablo 5.13: Lenet modeli Pytorch / Brevitas kodu.

Pytorch Lenet Model Kodu	Brevitas Lenet Model Kodu
<pre>class Lenet(nn.Module): def __init__(self): super().__init__() self.layers = nn.Sequential(self.conv1 = nn.Conv2d(1, 6, 5) self.conv2 = nn.Conv2d(6, 16, 5) self.fc1 = nn.Linear(16 * 4 * 4, 120) self.fc2 = nn.Linear(120, 84) self.fc3 = nn.Linear(84, 10)) def forward(self, x): return self.layers(x)</pre>	<pre>class Lenet(Module): super(Lenet, self).__init__() self.linear_features = ModuleList() self.conv_features = ModuleList() self.conv_features.append(QuantIdentity(QTYPE,1)) self.conv_features.append(QuantConv2d(5 , 1,16,1,QTYPE)) self.conv_features.append(BatchNorm2d(16, eps=1e-4)) self.conv_features.append(QuantIdentity(ActQuant,1)) self.conv_features.append(MaxPool2d(k=2)) self.conv_features.append(QuantConv2d(5 , 16,6,1,QTYPE)) self.conv_features.append(BatchNorm2d(6, eps=1e-4)) self.conv_features.append(QuantIdentity(ActQuant,1)) self.conv_features.append(MaxPool2d(k=2)) self.linear_features.append(QuantLinear(256,120,1,QTYPE)) self.linear_features.append(QuantLinear(120,84,1,QTYPE)) self.linear_features.append(BatchNorm1d(in_features)) self.linear_features.append(QuantIdentity(QTYPE,1)) self.linear_features.append(Dropout(p=DROPOUT)) self.linear_features.append(QuantLinear(84,10,1,QTYPE)) self.linear_features.append(TensorNorm()) def forward(self, x): x = 2.0 * x - torch.tensor([1.0], device=x.device) for mod in self.conv_features: x = mod(x) x = x.view(x.shape[0], -1) for mod in self.linear_features: x = mod(x) return x</pre>



Şekil 5.27: Lenet-5 modeli blok şeması.

Çalışmanın bu aşamasında MLP model için tasarlanan hızlandırıcılarla kıyaslama yapabilmek için W1A1, W1A2 ve W2A2 niceleme parametreleri ile Brevitas kütüphanesi kullanılarak Lenet modelleri oluşturuldu. Bu modeller 100 eğitim turu ile eğitildi ve bu eğitim aşamasında her turun doğruluk ve kayıp verileri kaydedilerek aşağıdaki grafik oluşturuldu. Niceleme türü 1 bit seçildiğinde her iki veri setinin eğitiminde de kayıp, doğruluk grafiklerinde sıçramalar oluşmakta ve öğrenme oranının diğerlerine kıyasla daha düşük olduğu görülmektedir. MNIST veri setinde her üç niceleme türünde de %95 üzeri doğruluk oranı elde edilirken, FashionMNIST veri setinde bu oran yaklaşık %85'ler civarındadır.

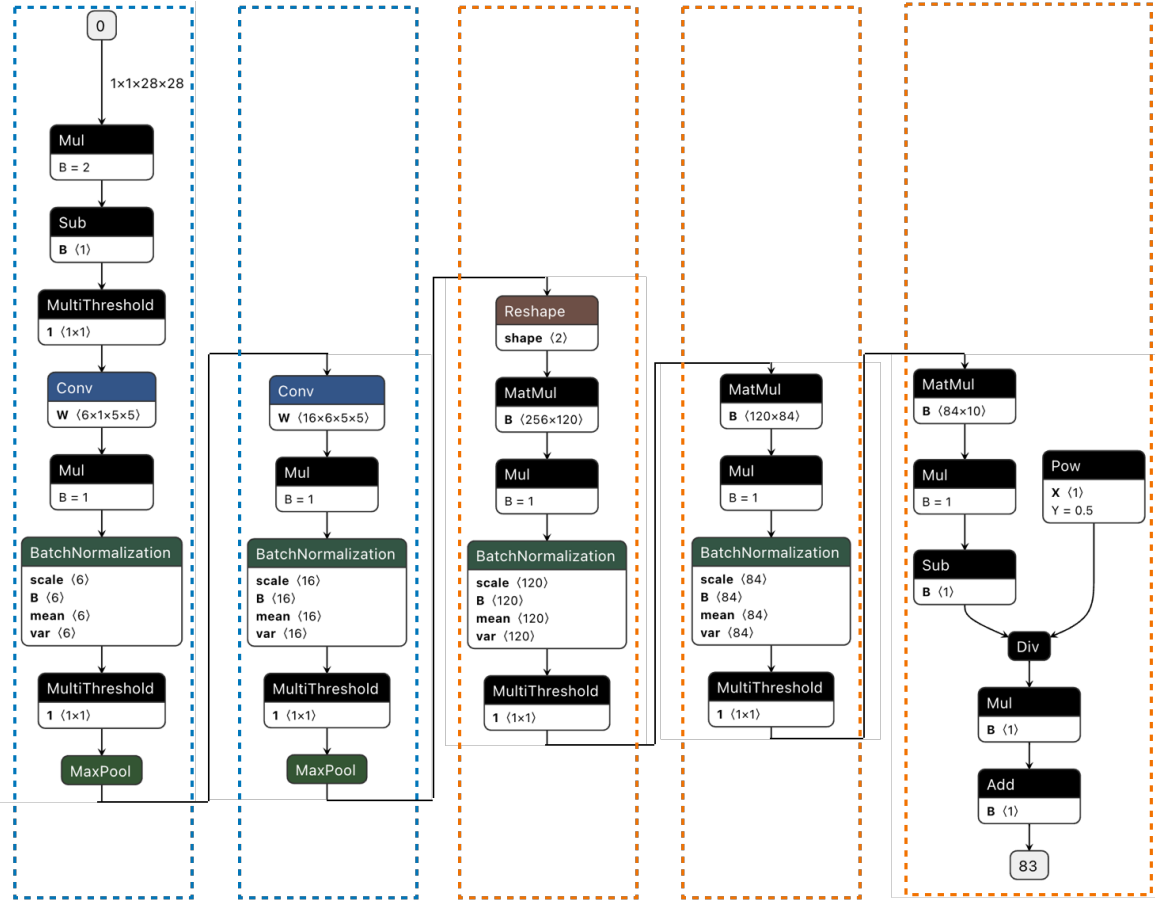


Şekil 5.28: Lenet modeli üç farklı hızlandırıcınının eğitim aşaması doğruluk ve kayıp grafiği.

5.2.2.2 Lenet Modeli FINN Dönüşümleri

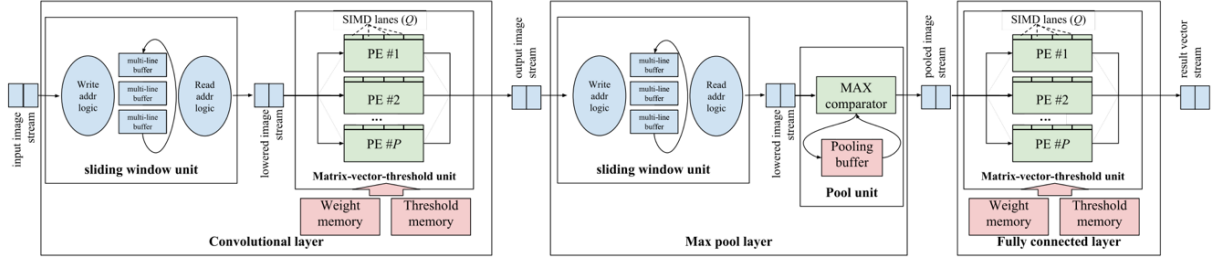
Önceden eğitmiş olduğumuz Lenet modeli FINN Brevitas dönüştürücü ile ONNX biçimine dönüştürülerek, ağırlık ve parametrelerle birlikte dışa aktarıldı. Eğitim sırasında kaydedilen

en iyi eğitim verileri FINN çerçevesine aktarılarak dönüşümlere başlanmıştır. İçerik aktarılan modelin Netron programındaki görseli Şekil 5.29'daki gibidir.



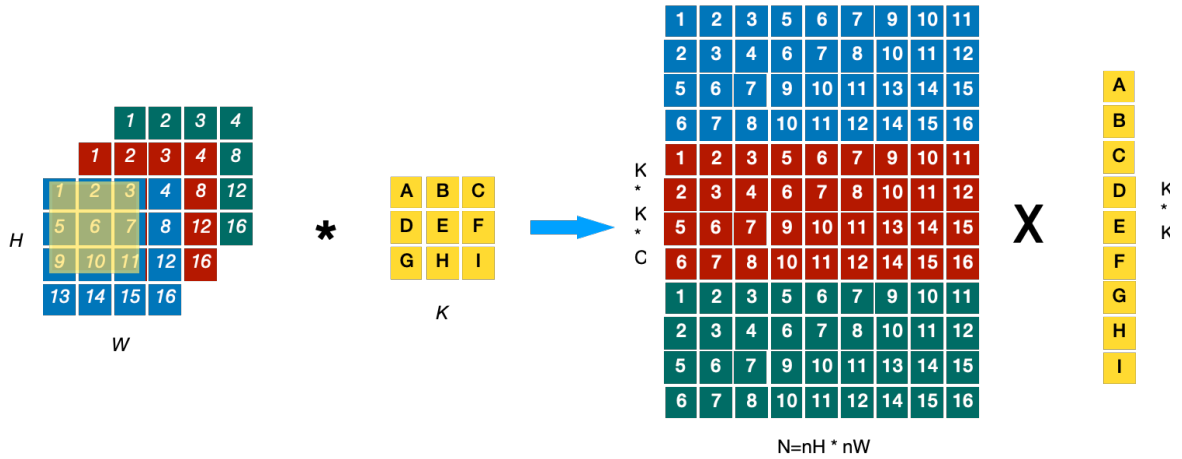
Şekil 5.29: Brevitas ile eğitilmiş Lenet modeli onnx görüntüsü.

Lenet modeli iki evrişim ve maksimum havuzlama ile üç tam bağlantılı katmandan meydana gelmektedir. Model Brevitas kütüphanesi kullanılarak nicelemeli olarak yukarıdaki gibi oluşturulmuştur. İlk mavi blok 6 çıkış kanalı olan evrişim ve havuz katmanını, diğer mavi blok ise 16 çıkış kanalı olan evrişim ve havuz katmanını meydana getirir. Ardından gelen sarı bloklar ise sırasıyla 120-84-10 çıkışlı tam bağlantı katmanlarını oluşturmaktadır. FINN çerçevesine yüklenen modele sırasıyla TidyUp, PrePost, ve StreamLine dönüşümleri uygulanmıştır. Bu dönüşümlerin işlevi MLP modeli oluşturulurken izah edildiği için tekrar edilmemiştir. Evrişim, havuzlama ve tam bağlantı katmanlarında veri akışı FINN-R isimli makalede Şekil 5.30'da verilen şekil ile görselleştirilmiştir [51].



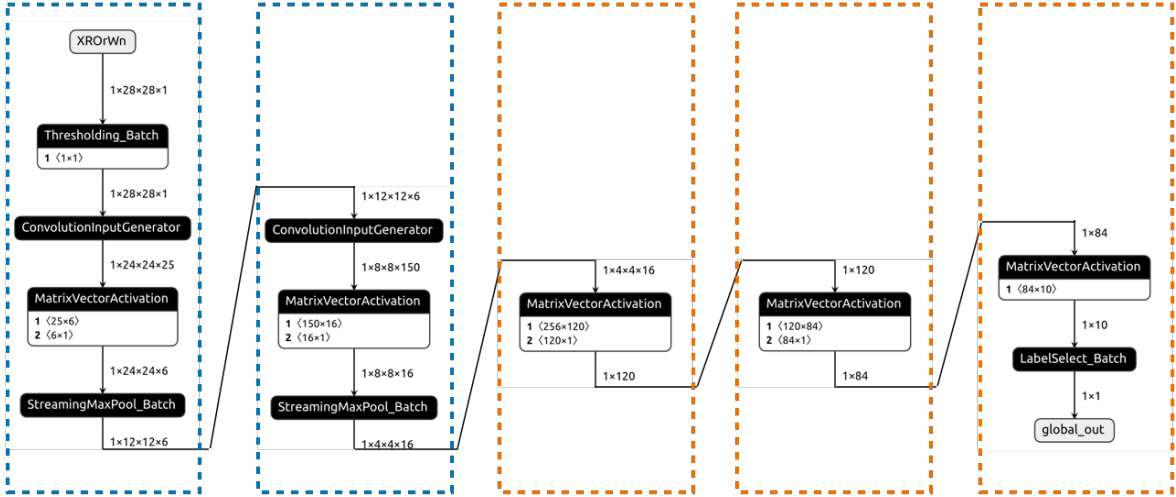
Şekil 5.30: Evrişim, havuzlama ve tam bağlantı katmanlarında veri akışı [51].

StreamLine dönüşümünden sonra LowerConvstoMatMul dönüşümü kullanılır. Bu dönüşüm evrişim düğümlerini Im2Col fonksiyonunu kullanarak kayan pencere dizilerine dönüştürür. Böylece evrişim işlemi yapmak yerine basit matris çarpımı işlemi kullanılır. Bit akışı modelinde giriş matrisinin her satır çekirdek matrisin düzleştirilmiş biçimi ile karşılıklı olarak çarpılır ve paralel toplayıcılarla toplanarak özellik matrisi elde edilir.



Şekil 5.31: Evrişim işleminin Im2Col işlemine dönüştürülmesi.

Tasarlanan ağda nicelleştirme parametresi 1 bit kullanıldıysa ConvertBipolarMatMulToXnorPopcount dönüşümü kullanarak ağın XNOR lojik kapıları ve bit sayaçları (popcount) ile oluşturulmasını sağlayabiliriz. Bu dönüşümler sonucu modeldeki evrişim düğümlerinin (Conv), Im2Col ve MatMul düğümlerine dönüştüğünü görebiliriz. Ardından uygulanan Streamline dönüşümü, çıkarma (Sub) düğümlerini toplamaya, bölme (DIV) düğümlerini çarpmaya, normalizasyon (BatchNorm) düğümlerini ise çarpma ve toplama düğümlerine dönüştürür. Ayrıca BatchNorm, Mul, Add gibi düğümler MultiThreshold düğümleriyle değiştirilmiştir.

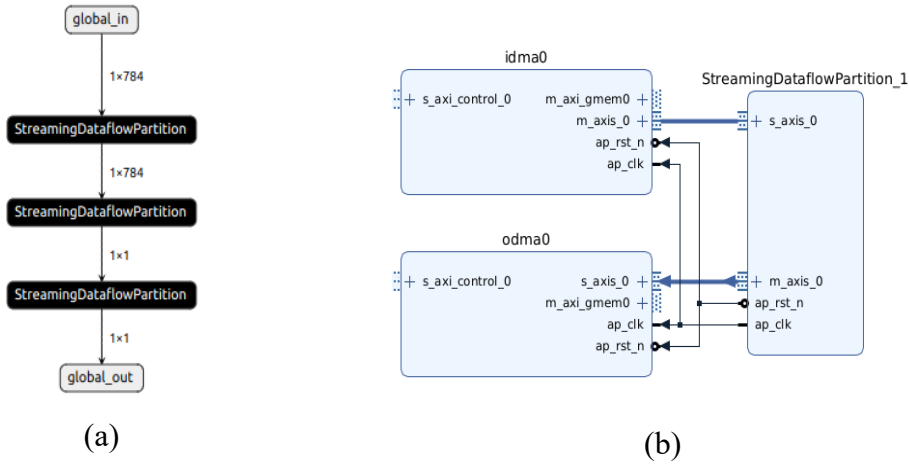


Şekil 5.32: StreamLine ve katlama parametrelerinin eklendiği dönüşüm sonrası modelin görünümü.

Modele StreamLine dönüşümünün ardından finn-hlslib kütüphanesi kullanılarak düğümler HLS katmanlarına dönüştürülür. Bu dönüşümlerle niceleme türüne göre Matmul ya da XnorPopcountMatMul katmanları ve MultiTreshold katmanları, MatrixVectorActivation katmanlarına dönüşür. FINN model sarmalayıcı ile açılan düğümlere PE ve SIMD katlama parametreleri eklenerek hızlandırıcı paralelleştirilir. Modele katlama parametrelerinin nasıl ekleneceği MLP model kısmında aktarılmıştır.

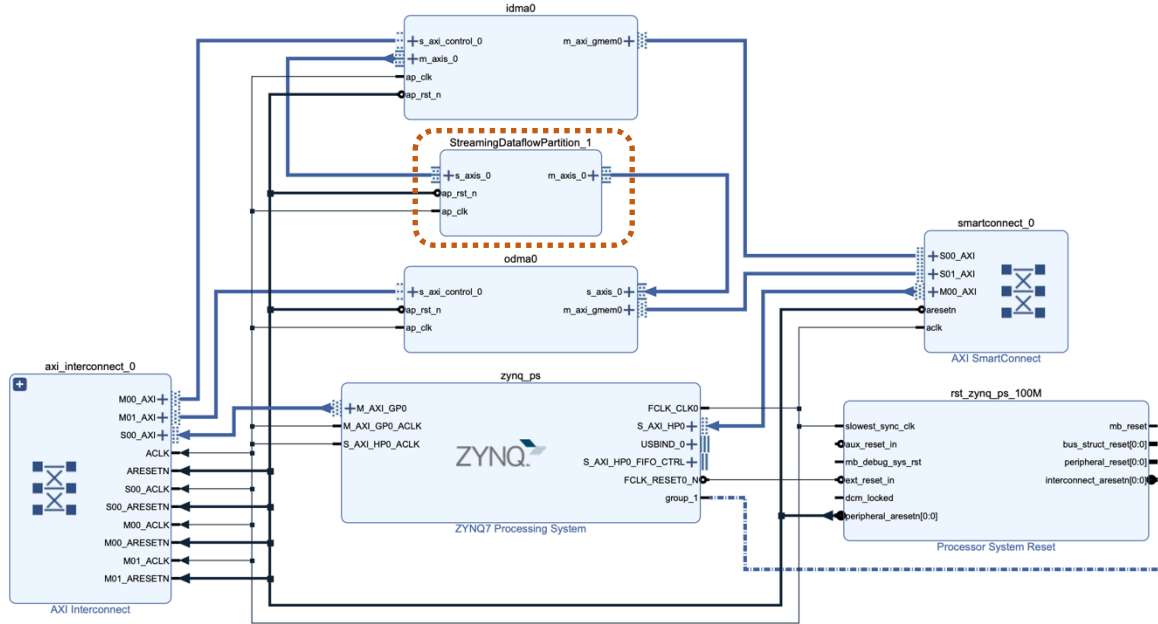
5.2.2.3 FINN Lenet Hızlandırıcı Donanımı Oluşturma

FINN donanım oluşturma aşamasında MLP hızlandırıcıda olduğu gibi ZynqBuild modülü kullanılır. Model sarmalayıcı ile açılan modele platform ve periyod bilgileri parametre olarak gönderilir ve Xilinx Vivado programı üzerinde hızlandırıcı donanımı oluşturulur. Lenet modeli ve oluşturulan hızlandırıcı donanımı Şekil 5.33'te görülmektedir.



Şekil 5.33:(a) Lenet veri akışı modeli (b) Lenet hızlandırıcı IP bloğu.

Şekil 5.33 (a)'da modelin son halinin Onnx formatında görüntüsü, Şekil 5.13 (b)'de ise Vivado ile oluşturulan IP blokları görülmektedir. Her iki resimde de ortada bulunan blok Lenet hızlandırıcı bloğudur. Üstte giriş hafıza erişimi bloğu altta ise çıkış hafıza erişim bloğu bulunur. Bu iki blok hızlandırıcı ile DRAM arasında veri taşıyan bölümlerdir. Lenet hızlandırıcı donanımı ZYNQ işletim sistemi ile AXI ara yüzleri kullanılarak Şekil 5.34'te görüldüğü gibi birbirine bağlanır. SoC özelliği ile PYNQ üzerinde bulunan Arm işlemci giriş verilerini boru hattı olarak hızlandırıcıya aktarır ve çıkış biriminden geri alır.



Şekil 5.34: Lenet hızlandırıcı ve ZYNQ işletim sistemi bağlantısı.

5.2.2.4 Lenet Hızlandırıcı PYNQ Üzerinde Uygulama

SSH bağlantısı kullanılarak PYNQ sürücüsü ve tasarlanan hızlandırıcı PYNQ Z1 kartına aktarılır. Yine SSH bağlantısı ile fashion.npz ya da mnist.npz veri seti dosyaları ile hızlandırıcı test edilir. Lenet hızlandırıcı W1A1, W1A2 ve W2A2 niceleme türlerine göre hem MNIST hem de FashionMNIST veri setleri ile test edilmiştir. Testlerde hızlandırıcının niceleme türlerine göre kaynak kullanımı ve verimliliği analiz edilmiştir.

Tablo 5.14: Lenet hızlandırıcı donanım test planı.

Katlama	Niceleme		
	W1A1	W1A2	W2A2
L	Donanım1	Donanım2	Donanım3
H	Donanım4	Donanım5	Donanım6

5.2.2.5 Lenet Hızlandırıcı Uygulaması Bulguları

Uygulamanın bu aşamasında PE ve SIMD parametreleri MLP modeli için aşağıdaki tabloda belirtildiği gibi toplam katlama 128 ve 1 olacak şekilde iki farklı katlama türü belirlenmiştir. Her iki katlama türü W1A1, W1A2 VE W2A2 niceleme oranları ile eğitilen modellere uygulanarak hızlandırma sonuçları ve kaynak kullanımı analiz edilmiştir.

Tablo 5.15: Lenet modeli iki farklı katlama konfigürasyonu.

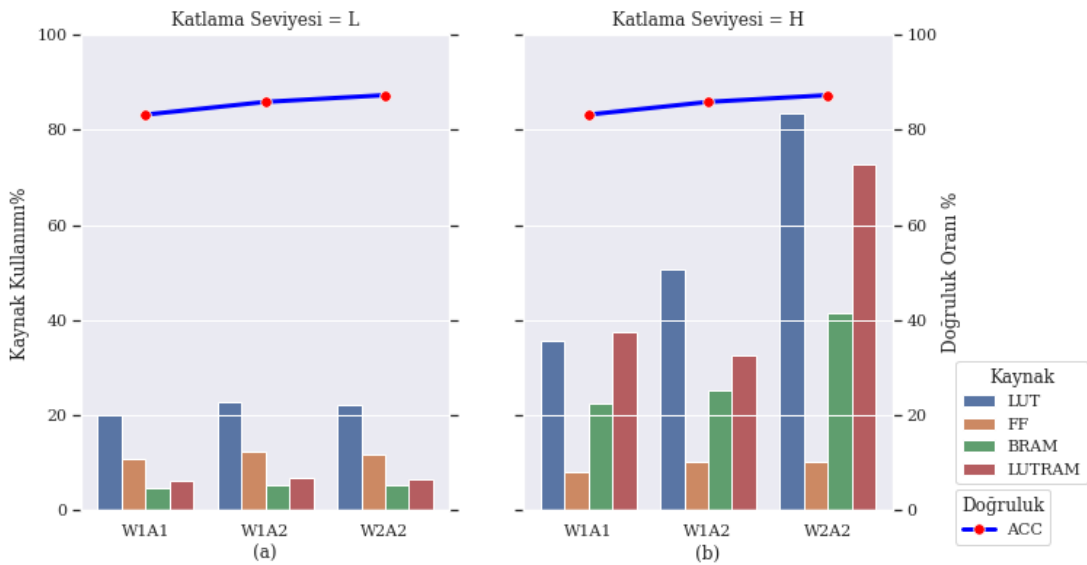
Konfigürasyon	H	L
Katman İsmi	CNV-CNV-FC-FC-FC	CNV-CNV-FC-FC-FC
Katman Çıkış	6-16-120-84-10	6-16-120-84-10
PE	6-16-60-42-5	1-1-1-1-1
Katman Giriş	1-6-256-120-84	1-6-256-120-84
SIMD	1-6-8-60-1	1-1-1-1-1
In_fifo	128-128-128-128-3	16-64-64

Yukarıdaki tabloda belirtildiği gibi iki farklı katlama konfigürasyonu belirlenmiştir. Yüksek katlama seviyesi olan konfigürasyon H ile düşük katlama seviyeli konfigürasyon ise L olarak kodlanmıştır. Bu iki konfigürasyon ve üç niceleme çeşidine göre FashionMNIST veri seti ile eğitilen modeller için altı farklı hızlandırıcı donanımı tasarlanmış ve test edilmiştir. Bu hızlandırıcıların FPGA kaynak kullanımı ve 10000 görüntünün sistemden geçiş süreleri ile saniyede işleyebilecekleri resim sayısı Tablo 5.16’da verilmiştir.

Tablo 5.16: Lenet hızlandırıcı donanımları kaynak kullanımı ve verim tablosu.

Nicleme Modeli	W1A1		W1A2		W2A2	
	L	H	L	H	L	H
Konfigürasyon						
LUT	19,87%	35,77%	22,12%	28,12%	22,29%	83,43%
LUTRAM	6,13%	7,97%	6,51%	8,11%	6,56%	10,30%
FF	10,79%	22,40%	14,64%	20,11%	11,87%	41,45%
BRAM	4,64%	37,50%	5,36%	9,64%	5,36%	72,86%
BUFG	20,00%	20,00%	20,00%	20,00%	20,00%	20,00%
Süre (ms)	18677	1467	18672	1468	18674	1467
Geçiş (Resim/sn)	535	6812	535	6809	535	6809

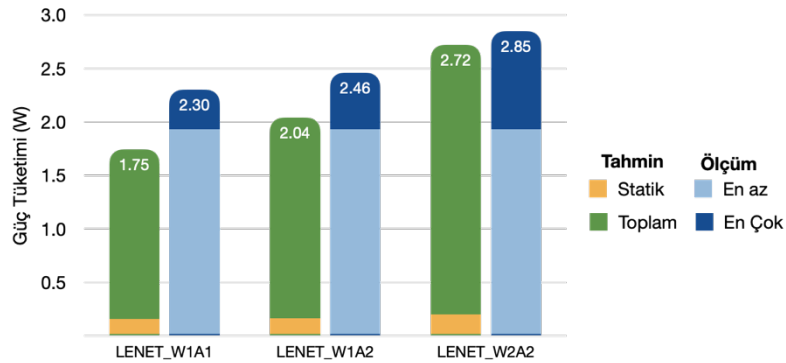
Tablodaki veriler incelendiğinde 10000 görüntünün testinin işleme süresinin nicleme bitlerine değil katlama faktörlerine bağlı olduğunu göstermektedir. Hemen hemen 3 nicleme türünde de L konfigürasyonunda saniyede 535 resim işlenebilirken H katlama konfigürasyonunda bu sayı 6800'lere çıkmaktadır. Bu da Lenet modelin H katlama faktörü ile yaklaşık 12 kat hızlandırıldığını göstermektedir. H katlama faktörlü hızlandırıcılar 10000 görüntüyü yaklaşık 1400 ms'de işlerken, L katlama konfigürasyonlu hızlandırıcı modelleri yaklaşık 18600 ms'de işleyebilmektedir. Hızlandırıcıların kaynak kullanım oranları ise hem nicleme bit genişliklerine hem de katlama faktörüne bağlıdır. Çalışmanın bu kısmında yapılan testlerde katlama konfigürasyonları evrişim ve tam bağlantılı katmanların her ikisi içinde yapılmıştır. Sonraki test modellerinde katlama faktörleri belirlenirken evrişim ve tam bağlantılı katmanların katlama faktörleri ayrı ayrı belirlenerek farklı kombinasyonlar oluşturulmuştur.



Şekil 5.35: Lenet hızlandırıcı nicleme ve katlama seviyelerine göre kaynak kullanım grafiği.

Lenet modeli için tasarlanan hızlandırıcının farklı niceleme türleri ve iki farklı katlama konfigürasyonu için kaynak kullanım oranları grafikte aktarılmıştır. Lenet modeli için oluşturulan altı hızlandırıcının doğruluk oranları çizgi grafikle kaynak kullanımları ise bar grafikte görselleştirilmiştir. a ve b grafiğinde aynı çizginin olduğu görülmektedir. Bu durum katlama seviyelerinin doğruluk oranını değiştirmedeğini göstermektedir. Bu sonuç katlama seviyesinin derin öğrenme hızlandırıcısının doğruluk seviyesini etkilemediğini göstermektedir. Solda bulunan grafik L katlama seviyesindeki hızlandırıcılar incelendiğinde niceleme seviyesi arttıkça LUT ve FF kullanım oranları artarken bellek kaynak kullanımının değişmediği görülmektedir. Katlama seviyesi H olan hızlandırıcılarda bellek ve mantık kaynaklarının kullanım oranı niceleme seviyesi ile doğru orantılıdır. H katlama seviyeli hızlandırıcılarda W1A1 ile W1A2 modellerinde bellek kullanımı aynı seviyelerde iken W2A2 hızlandırıcıda bellek kullanımı yaklaşık iki katına çıkmıştır.

Niceleme seviyelerine göre tasarlanmış olduğumuz bu üç hızlandırıcı donanımının güç analizleri, hem USB güç ölçerle fiziksel olarak hem de Vivado ile hesaplanan güç tüketimleri açısından değerlendirilmiştir. Alınan sonuçlar fiziksel ölçümde en az en çok, tahmin uygulamasında ise statik ve toplam olmak üzere gruplandırılarak Şekil 5.36'da görselleştirilmiştir.



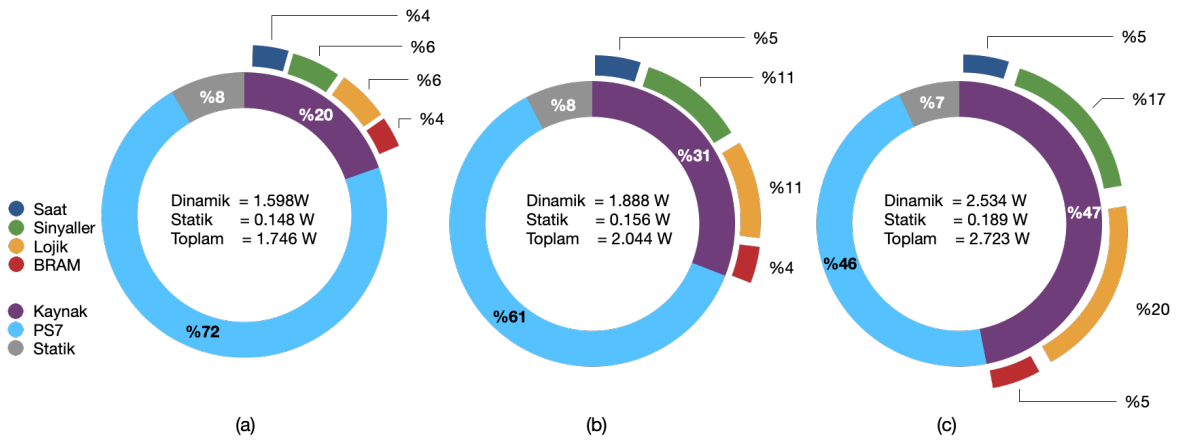
Şekil 5.36: Niceleme seviyelerine göre Lenet modeli hızlandırıcıların güç tüketimi ölçüm ve tahmin sonuçları grafiği.

Lenet modeli için geliştirilen hızlandırıcıların Vivado güç tahmin uygulaması ile hesaplanan güç tüketimleri daha detaylı olarak incelenebilmesi için kaynak türü ve tüketim yerine göre gruplandırılarak Tablo 5.17'de listelenmiştir.

Tablo 5.17: Lenet modeli hızlandırıcılar Vivado ile hesaplanan güç tüketimleri.

Tüketim Türü	Lenet W1A1	LENET W1A2	LENET W2A2	Tüketim Yeri
Dinamik	0.073	0.101	0.135	Saat
	0.101	0.234	0.475	Sinyaller
	0.106	0.216	0.524	Lojik
	0.063	0.081	0.144	BRAM
	0.342	0.632	1.278	Toplam Kaynak
	1.256	1.256	1.256	PS7
Statik	0.148	0.156	0.189	
Toplam	1.746	2.044	2.723	

Hızlandırıcıların güç tüketimleri tüketim türüne göre dinamik ve statik olarak iki gruba ayrılırken, dinamik tüketimler tüketim yerine göre beş bileşene ayrılmıştır. Bu bileşenlerden ilk dördü hızlandırıcının kullandığı kaynaklar saat, sinyaller, lojik elemanlar ve hafıza birimlerini, sonuncusu ise PS7 işletim sisteminin tükettiği gücü göstermektedir. Bu tablodaki güç tüketimleri Şekil 5.37’de görselleştirilmiştir.

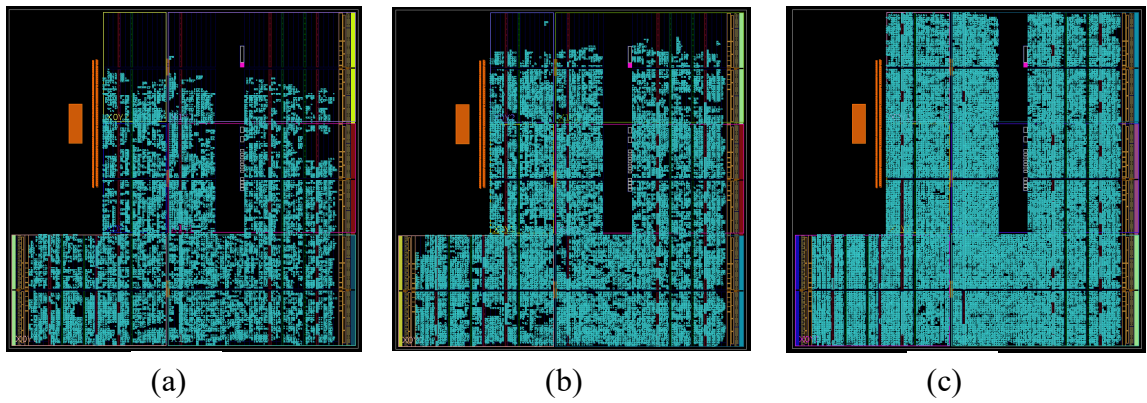


Şekil 5.37: Lenet modeli hızlandırıcılar (a) W1A1, (b) W1A2 ve (c) W2A2 niceleme seviyelerine göre güç analizi.

Üç farklı niceleme seviyesine göre yapılan analizler, niceleme seviyesinin artmasının toplam güç tüketimini artırdığını göstermektedir. Grafiklerde iç halkada Statik tüketim gri renkli bölümü, mor ve açık mavi bölüm ise PS7 ve kaynakların güç tüketim oranlarını göstermektedir. PS7 ve Kaynak kullanımı toplamı dinamik tüketimi vermektedir. Dinamik güç tüketimlerinde Pynq Z1 kartını kontrol eden PS7 işletim sisteminin güç tüketiminin diğer donanımlardan daha yüksek olduğu görülmektedir. Tasarlanan hızlandırıcı

donanımlarının güç tüketimi üstündeki etkisi dış halkada bulunan dört bileşenle iç halkadaki mor bölümün dağılımını ifade eder. Lenet W1A1 hızlandırıcıda toplam kaynak tüketimi %20'ler de iken W1A2 modelinde bu oran %31, W2A2 modelinde ise %47 seviyesine çıkmıştır. En yük sek gücü tüketen W2A2 modeli 2.72 W enerji tüketmiştir. Bu tüketim CPU ya da GPU gibi donanımlarla kıyaslanamayacak kadar küçüktür.

Lenet modeli için tasarlanan hızlandırıcı donanımlarının Vivado uygulamasından alınan alan kullanımları Şekil 5:38'de verilmiştir.



Şekil 5.38:Lenet hızlandırıcı donanımları (a) W1A1, (b) W1A2 ve (c) W2A2 alan kullanımları.

Bu aşamaya kadar olan çalışmada niceleme ve katlama parametrelerinin hızlandırıcı donanımının kaynak kullanımı ve verimi üzerindeki etkileri, güç tüketimleri ve alan kullanımları incelendi. Tasarlamış olduğumuz MLP ve Lenet modeli hızlandırıcıların kaynak kullanımlarında tam bağlantı katmanı ve evrişim katmanlarının farklı etkileri olduğu görüldü. Bu nedenle Lenet modeli için evrişim ve tam bağlantılı katmanların katlama seviyelerini ayrı ayrı değiştirerek yeni bir hızlandırıcı grubu planlandı. Evrişim ve tam bağlantı katmanlarında PE-SIMD katlama parametrelerinin kaynak kullanımını ve verimi nasıl etkilediğini daha iyi ölçebilmek adına aşağıdaki tabloda verilen test planı tasarlanmıştır. Bu testlerde W1A2 niceleme ile eğitilen Lenet modeli FINN çerçevesine alınarak evrişim katmaları için L, M, H ve tam bağlantı katmanları için L, M, H olmak üzere 3'er farklı konfigürasyonun kombinasyonları ile 9 farklı derin öğrenme hızlandırıcısı tasarlandı. Bu dokuz hızlandırıcı donanımının hepsinin katlama parametrelerinde FIFO değerleri sabit bırakılmamıştır. Belirlenen katlama konfigürasyonları aşağıdaki tabloda görülmektedir. Tablo 5.18'de tasarlanan hızlandırıcıların Lenet model katmanlarının giriş ve

çıkış kanal sayıları (*ilk iki satırda*) ile her katman için PE ve SIMD değerlerinin kombinasyonları olan 9 farklı konfigürasyon görülmektedir.

Tablo 5.18: Evrişim ve tam bağlantılı katmanlar için hızlandırıcı katlama konfigürasyonu.

Konfigürasyon	Katman	Cnv1	CNV2	FC1	FC2	FC3	Katlama	
							CNV	FC
	Giriş	1	6	256	120	84		
	Çıkış	6	16	120	84	10		
Konf.1	PE	1	1	1	1	1	L	L
	SIMD	1	1	1	1	1		
Konf.2	PE	1	1	8	4	5	L	M
	SIMD	1	1	8	8	1		
Konf.3	PE	1	1	60	42	5	L	H
	SIMD	1	1	8	60	1		
Konf.4	PE	3	8	1	1	1	M	L
	SIMD	1	3	1	1	1		
Konf.5	PE	3	8	8	4	5	M	M
	SIMD	1	3	8	8	1		
Konf.6	PE	3	8	60	42	5	M	H
	SIMD	1	3	8	60	1		
Konf.7	PE	6	16	1	1	1	H	L
	SIMD	1	6	1	1	1		
Konf.8	PE	6	16	8	4	5	H	M
	SIMD	1	6	8	8	1		
Konf.9	PE	6	16	60	42	5	H	H
	SIMD	1	6	8	60	1		

Konfigürasyon-1 katlama seviyesi en düşük olan konfigürasyondur. Hem evrişim hem de tam bağlantı katmanları için PE ve SIMD değerleri bir verilmiştir ve bu katlama seviyesi L olarak kodlanmıştır. Evrişim katmanları için PE (3,8) ve SIMD (1,3) değeri M olarak kodlanırken, tam bağlantılı katmanlarda M seviyesi PE (8,4,5) ve SIMD (8,8,1) olarak kodlanmıştır. H seviyeleri ise evrişim katmanlarında PE (6,16) – SIMD (1,6), tam bağlantı katmanlarında ise PE (60,42,5) ile SIMD (8,60,1) tanımlanmıştır. FIFO belek parametreleri tüm konfigürasyonlarda aynı kalmış, değiştirilmemiştir.

Brevitas ile W1A2 niceleme türünde oluşturulup eğitilen Lenet modeli onnx formatına dönüştürülerek FINN çerçevesine dahil edilmiştir. FINN çerçevesi kullanılarak yukarıdaki konfigürasyonlarda 9 adet hızlandırıcı donanım oluşturuldu. Bu donanımlar FashionMNIST veri setindeki 10000 test görüntüsü ile test edildi. Yapılan testlerde kaynak kullanımları ile doğruluk, toplam geçiş süresi (10000 görüntü için) ve saniyede işlenen görüntü sayıları Tablo 5.19’da görünmektedir.

Tablo 5.19: Tasarlanan Lenet hızlandırıcılarının kaynak verim tablosu.

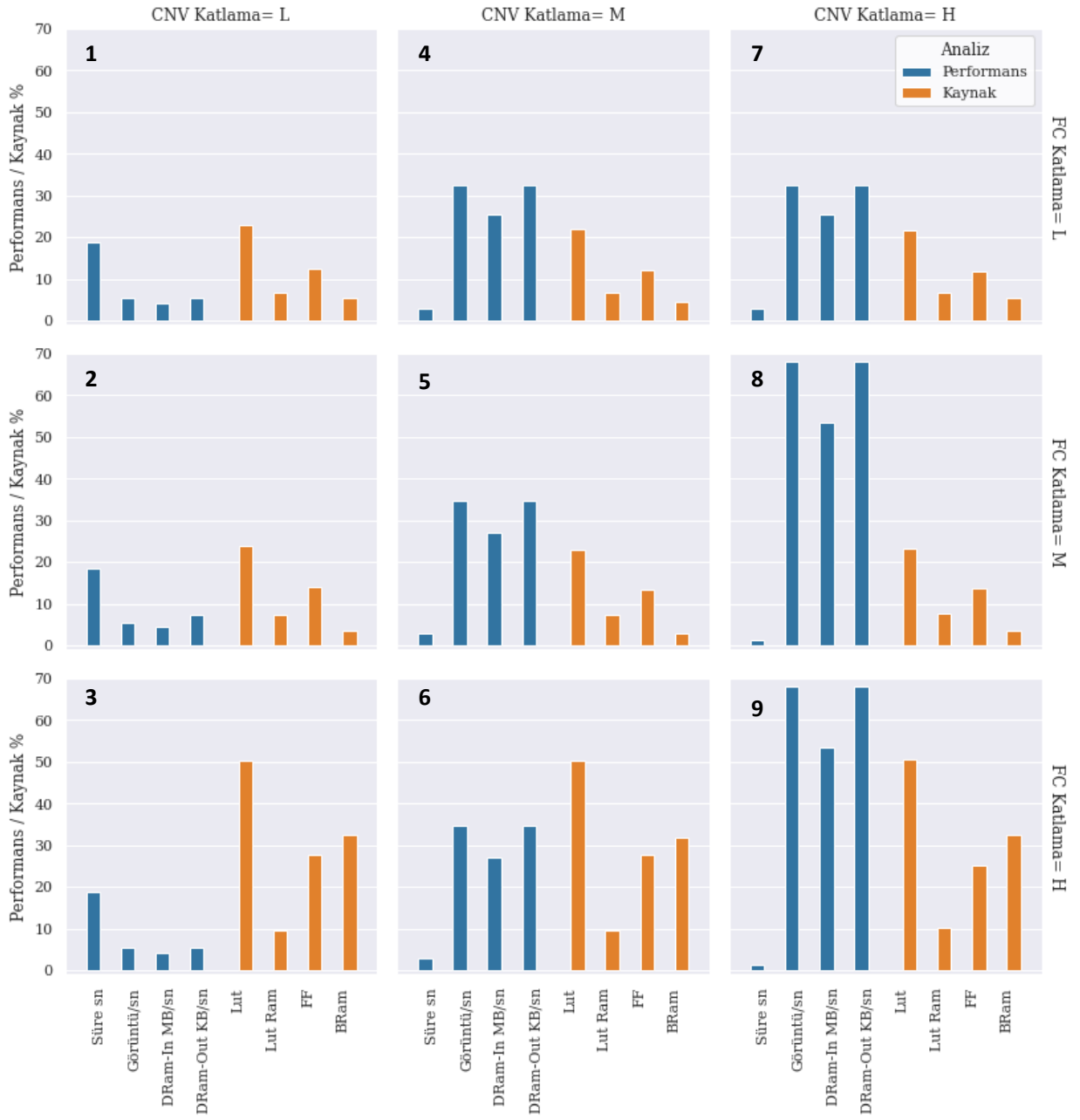
Parametre	Konf.-1	Konf.-2	Konf.-3	Konf.-4	Konf.-5	Konf.-6	Konf.-7	Konf.-8	Konf.-9
CNV Katlama	L	L	L	M	M	M	H	H	H
FC Katlama	L	M	H	L	M	H	L	M	H
Toplam Süre[ms]	18672	18455	18672	3073	2880	2880	3073	1468	1468
Verim[görüntü/s]	535	542	535	3253	3471	3471	3253	6809	6809
DRAM-in BW [MB/s]	0,42	0,44	0,42	2,55	2,72	2,72	2,55	5,34	5,34
DRAM-Out BW [KB/s]	0,53	0,74	0,53	3,25	3,47	3,47	3,25	6,80	6,80
LUT	12165	12677	26829	11636	12141	26692	11481	12396	26935
LUTRAM	1183	1294	1669	1149	1262	1677	1197	1351	1767
FF	13271	14779	29352	12813	14321	29574	12462	14499	26752
BRAM	7,5	5	45,5	6,5	4	44,5	7,5	5	45,5

Katlama seviyelerine göre oluşturulan ve test edilen hızlandırıcıların kaynak kullanımları PYNQ Z1 FPGA platformunda bulunan kaynakları kullanma oranına göre yüzdelik olarak kullanılmıştır. Aynı grafikte verim değerlerini de inceleyebilmek için bu değerler 0-100 arasında olacak şekilde Tablo 5.20’de belirtilen katsayılarla çarpılarak normleştirilmiştir.

Tablo 5.20: Normleştirme katsayıları.

Toplam Süre[ms]	Verim [görüntü/s]	DRAM-in [MB/s]	DRAM-Out [KB/s]
10^3	10^{-2}	10	10

Tablo 5.19’da belirtilen değerler normleştirildikten sonra 3 sütun ve 3 satırda tüm katlama konfigürasyonları görünecek şekilde aşağıdaki grafikte görselleştirilmiştir (Şekil 5.39). Sütunlarda evrişim (CNV) katmanlarında katlama seviyesi sırasıyla soldan sağa doğru L, M ve H iken satırlarda tam bağlantılı katman (FC) katlama seviyeleri yukarıdan aşağıya doğru L,M ve H olarak belirlenmiştir.

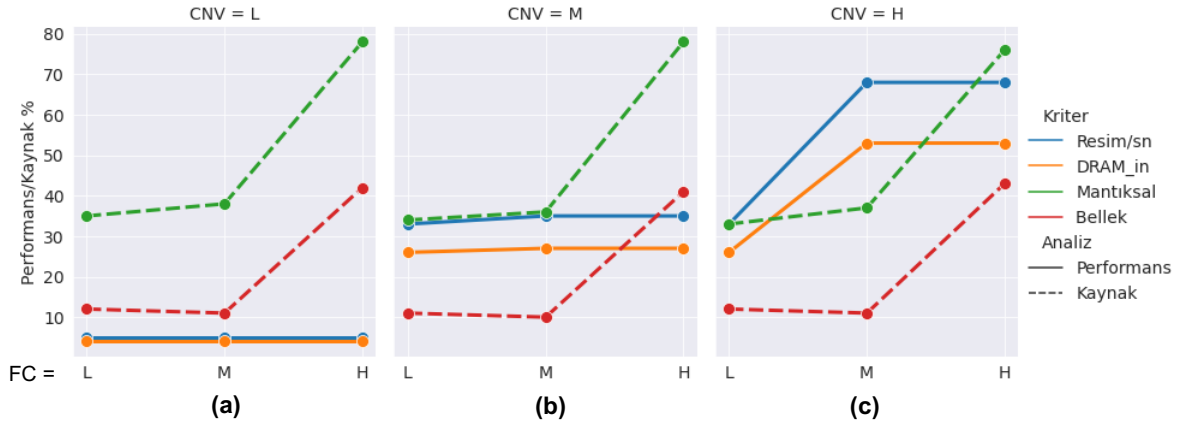


Şekil 5.39: Evrişim ve tam bağlantılı katman katlama faktörlerinin hızlandırıcılar üstünde kaynak kullanımı ve verime etki grafiği (Performans/Kaynak kullanımı açısından en iyi konfigürasyon 8 numaralı konfigürasyondur).

Yukarıdaki grafikler konfigürasyon türüne göre 1-9 arasında numaralandırılmış, kaynak ve performans açısından gruplandırılmıştır. Mavi grup performans verilerini turuncu grup ise FPGA kaynaklarını göstermektedir. Performans bloğundaki ilk değer 10000 test görüntüsünün hızlandırıcıda işleme süresini verirken yanındaki değer matematiksel olarak bu değerın ters orantısı alınarak elde edilen saniyede işlenebilecek görüntü sayısını vermektedir. Süre değeri diğer performans kriterleri ile ters orantılı olduğu için bu değer

gurubu elde edilerek grafiğe eklenmiştir. Grafiklerde en yüksek verim 8 ve 9 numaralı grafiklerde olduğu görülmektedir. Bu iki konfigürasyonda da hızlandırıcının saniyede yaklaşık 6800 resim işleyebildiği görülmektedir. Ayrıca DRAM giriş ve çıkış bant genişliği en yüksek seviyededir. Konfigürasyon-8’de evrişim katmanları katlama seviyesi H, tam bağlantılı katman katlama seviyesi ise M iken, konfigürasyon-9’da HH olarak belirlenmiştir. Bu katmanlarda verim aynı olmasına rağmen kaynak tüketimi açısından büyük fark görülmektedir. Konfigürasyon-9 hem LUT ve FF gibi mantıksal kaynaklarda hem de LUTRAM ve BRAM gibi hafıza kaynaklarının kullanımında konfigürasyon-8’den yaklaşık iki kat daha fazla kaynak tüketmiştir. Bu iki katman açıkça gösteriyor ki tam bağlantılı katmanlarda katlama oranının artması kaynak kullanımını artırmasına rağmen belirli bir seviyeden sonra performansa katkı sağlamamaktadır. Bu durum hafıza birimlerinin veri akışındaki darboğazdan kaynaklanmaktadır. En düşük katlama oranına sahip konfigürasyon-1 ile en az kaynakla en iyi performansı gösteren konfigürasyon-8 karşılaştırıldığında 10000 test görüntüsünün işleme süreleri konfigürasyon-1’de 18672 ms iken konfigürasyon-8’de 1468 ms seviyelerine düşmüştür. Bu da W1A2 seviyesinde nicelenmiş Lenet modeli için tasarlanan hızlandırıcının katlama metodu kullanılarak 12 kat hızlandırıldığını göstermektedir. Konfigürasyonları tam bağlantılı katman katlamaları sabit olacak şekilde evrişim katmanı katlamaları için (1-4-7), (2-5-8) ve (3-6-9) gruplarının verim değerleri açısından incelendi. Evrişim katmanında yapılan katlama matris çarpımı gibi yoğun işlemleri daha paralel hale getirmesinden dolayı hızlandırıcı performansına büyük katkı sağlamaktadır. Konfigürasyon-7 ve konfigürasyon-8 incelendiğinde her ikisinde de evrişim katmanı kalama seviyesinin H olmasına rağmen konfigürasyon-8 de tam bağlantı katmanının M seviyesine gelmesi verimi 2 kat artırmıştır. Yukarıdaki hızlandırıcılarda verim/kaynak tüketimi açısından en kötü performansı konfigürasyon-3 göstermiştir. Çok fazla kaynak tüketmesine karşın verim hiç katlama yapılmayan konfigürasyon-1 seviyesindedir. Hızlandırıcı boru hattı modeli ile çalıştığından, hattın ilerleyen bölümlerinde yapılan hızlandırma girişteki yavaşlıktan etkileneceği için geçersiz olur.

Çalışmanın bu bölümünde yapılan hızlandırıcı testlerini farklı bir bakış açısından analiz edebilmek için evrişim ve tam bağlantı katlama seviyelerine göre aşağıdaki performans grafiği oluşturulmuştur.



Şekil 5.40: Mantıksal ve bellek kaynaklarının katlama seviyelerine göre performans grafikleri.

Bu grafikte LUT ve FF birimleri kullanım oranları mantıksal kaynak, LUTRAM ve BRAM kaynaklarının toplamları ise bellek kaynakları olarak gruplandırılmıştır (Şekil 5.40). Kaynak verileri kesikli çizgi ile, saniyede işlenen resim sayısı ve DRAM giriş bant genişliği düz çizgi ile gösterilmiştir. Her evrişim katlama seviyesine göre (L, H, M) tam bağlantılı katman katlama seviyelerindeki veriler üç farklı çizgi grafiğinde görselleştirilmiştir. Performans verileri bir önceki tabloda belirtildiği gibi 0-100 arasında normalleştirilmiştir. Şekil 5:40'ta verilen grafikte (a) bölümünde, evrişim katlama oranı L seviyesinde iken tam bağlantılı katmanda yapılan katlamanın performansa etkisi olmamasına karşın kaynak tüketimini artırdığı gözlemlenmiştir. Şekil 5:40 (b)'de ise evrişim katlama oranının M seviyesine çıkması performansı yaklaşık dört kat artırmıştır. Evrişim katlama seviyesi H olan Şekil 5:40 (c) grafiği incelendiğinde tam bağlantılı katmanda katlama oranı L seviyesinde tutulunca performansın artmadığı Şekil 5:40 (b) grafiği ile hemen hemen aynı seviyelerde kaldığı görülür. Bu durum bir veri akışı hızlandırıcıda çıkışta yavaş bir katman varsa girişteki katmanların hızlı olması çıkışı aynı seviyede hızlandırmaz. Çünkü işlemi bitiren katman diğer katmandaki işlemin bitmesini bekler. Ayrıca Şekil 5:40 (c)'de bulunan FC katman katlama seviyeleri M ve H durumları katman türlerindeki katlamanın hızlandırıcıya etkileri açısından çok önemli bir göstergedir. Bu iki durumda katlama seviyesinin artmasına karşın performansın aynı kaldığı görülür. Bunun nedeni tam bağlantılı katmanlarda veri akışının çalışma frekansı ve bellek birimlerinin bant genişliği ile sınırlı olmasıdır. Çalışmamızın bu bölümünde tasarladığımız doku hızlandırıcı konfigürasyonu arasında kaynak tüketimi ve performans açısından değerlendirdiğimizde en başarılı sonuç 8 numaralı

konfigürasyonda gerçekleşmiştir. Bu konfigürasyonda CNV katlama seviyesi H, FC katlama seviyesi H olarak belirlenmiştir. Konfigürasyon-8, saniyede işlediği görüntü sayısı (FPS) 6809 olurken, konfigürasyon-1 yaklaşık 555 Fps seviyelerindedir. Bu durum 8 numaralı hızlandırıcınının 1 numaralı hızlandırıcıdan yaklaşık 13 kat daha iyi performansla çalıştığını göstermektedir.

6. SONUÇLAR ve ÖNERİLER

Bu tez çalışmasında FPGA tabanlı derin öğrenme hızlandırıcıları üzerinde çalışılmış, önce HLS tasarım ile sonra FINN çerçevesi ve Brevitas kütüphanesi kullanılarak derin öğrenme hızlandırıcıları geliştirilmiştir. Bu çalışma boyunca Tablo 6.1’de belirtilen özelliklerde 23 farklı hızlandırıcı geliştirilmiş ve incelenmiştir.

Tablo 6.1: Geliştirilen FPGA tabanlı derin öğrenme hızlandırıcıları özellik tablosu.

No	Teknoloji	Çerçeve	Model	Niceleme	Katlama Seviyesi		
1	HLS	Caffe	MLP	W8A8	-		
2			Lenet-5	W8A8	-		
3	FINN	Pytorch/Brevitas	MLP	W1A1	L		
4			MLP	W1A1	H		
5			MLP	W1A2	L		
6			MLP	W1A2	H		
7			MLP	W2A2	L		
8			MLP	W2A2	H		
9			FINN	Pytorch/Brevitas	Lenet-5	W1A1	L
10					Lenet-5	W1A1	H
11	Lenet-5	W1A2			L		
12	Lenet-5	W1A2			H		
13	Lenet-5	W2A2			L		
14	Lenet-5	W2A2			H		
15	FINN	Pytorch/Brevitas			Lenet-5	W1A2	L-L
16					Lenet-5	W1A2	L-M
17			Lenet-5	W1A2	L-H		
18			Lenet-5	W1A2	M-L		
19			Lenet-5	W1A2	M-M		
20			Lenet-5	W1A2	M-H		
21			Lenet-5	W1A2	H-L		
22			Lenet-5	W1A2	H-M		
23			Lenet-5	W1A2	H-H		

Başlangıçta HLS tasarım ile FPGA üzerinde Caffe kütüphanesi ile eğitilmiş MLP ve Lenet ağlarını hızlandıran 8 bit nicelemeli 2 hızlandırıcı donanımı tasarlandı. FPGA üzerinde gerçekleştirilen Lenet modeli için tasarlanan hızlandırıcı donanımı 1,18 s, MLP için tasarlanan ise 0,26 s işlem süresi ölçülmüştür. Lenet modeli hızlandırıcı %5 oranında doğruluğu artırırken yaklaşık 9 kat daha yavaş çalışmıştır. FPGA üzerinde HLS tasarım ile gerçekleştirilen Lenet ve MLP modeli hızlandırıcı uygulamasının CPU’dan yaklaşık 15 kat daha hızlı olduğu görülmektedir.

Bu hızlandırıcılarla daha düşük seviyeli niceleme yapılamadığı için Pytorch derin öğrenme çerçevesi ve Brevitas kütüphanesi kullanılarak farklı niceleme seviyelerinde hızlandırıcılar geliştirildi, MNIST ve FashionMNIST veri setleri kullanılarak eğitildi. Bu modeller HLS uygulamasında olduğu gibi MLP ve Lenet modellerinin farklı niceleme türlerindeki kombinasyonlarından meydana gelmektedir. Başlangıçta MLP ve Lenet modelleri için W1A1, W1A2 ve W2A2 şeklinde ağırlık ve aktivasyon bit niceleme seviyesi ve her seviye için düşük (L), yüksek (H) olmak üzere iki katlama parametresi belirlendi. Belirlenen nicelenmiş modelin katlama parametrelerine göre kombinasyonları olan 12 adet hızlandırıcı donanım FINN çerçevesi kullanılarak geliştirildi. Geliştirilen hızlandırıcılar MNIST ve FashionMNIST veri setleri ile test edildi. Hızlandırıcılar saniyede işleyebildikleri görüntü sayısı, mantık - bellek kaynakları kullanımı, güç tüketimleri ve çip üzerinde kapladıkları alan açısından analiz edildiler. MNIST gibi kolay öğrenilebilen veri setlerinde MLP modeli ile tasarlanan hızlandırıcılar yeterli gelmekte iken FashionMnist, CIFAR-10 gibi veri setleri için evrişim katmanı içeren modeller daha iyi çalışmaktadır. MLP W1A1 hızlandırıcısının H katlama seviyesinde yaklaşık 951 Kfps, Lenet W1A1 hızlandırıcısının H katlama seviyesinde yaklaşık 6,9 Kfps değerine ulaştığı görülmektedir. Her iki hızlandırıcı da kendi gruplarında diğer modellere göre en az kaynak kullanarak en yüksek performans elde etmişlerdir. Bu tezde tasarlanan hızlandırıcı donanımlarının performans ve kaynak kullanım sonuçları benzer şekilde MNIST ve FashionMNIST kullanan nicelemeli diğer FPGA tabanlı çalışmalardan elde edilen sonuçlarla karşılaştırılmak üzere Tablo 6.2’de listelenmiştir.

Tablo 6.2: Benzer çalışmaların sonuçlarının karşılaştırılması.

Çalışma	Niceleme	Model	BRAM	LUTs	FF	DSP	Çık (MHz)	İşlem (GOPs)	Güç (W)	Verim (GOPs/W)
FP-BNN[54]	1 Bit	MLP	2210	182301	-	20	150	5904	26,2	225,3
Zhao R.[55]	1 Bit	VGG	86	25289	28197	3	143	207	4,7	44,0
FINN[28]	1 Bit	LFC	396	82988	-	-	200	9086	22,6	402
FINN-R[51]***	1 Bit	MLP-4	220	25358	-	-	100	974	2,5	390
Laius[56]	8 Bit	Lenet	619	9071	1681	916	-	44,9	0,6*	-
BinaryEYE[57]	1 Bit	MLP	124	88000	115000	-	200	116	13,8	8,4
FPGA-NHAP[58]	16 Bit	MLP	83,5	12218	10325	64	200	0,98	0,54*	-
FCA-BNN[59]	1 Bit	LFC	1384	152800	-	10	166	1932	5,7	338,9
Full-BNN[60]	1 Bit	CNV	127	30075	-	100	125	-	3,5	-
Bu çalışma	1 Bit	MLP**	25,5	18931	24871	-	100	1019	2,05	496

* Çip içi güç tüketimi

** MLP modeli, W1A1 niceleme türü ve H (64) katlama seviyesinde geliştirilen donanım.

*** PYNQ-Z1 üzerinde yapmış oldukları çalışma sonuçları.

- Erişilemeyen veriler (-) ile işaretlenmiştir.

Geliştirmiş olduğumuz MLP hızlandırıcı kullanmış olduğumuz her iki veri seti içinde en iyi işlem/güç performansını göstermiştir. Umuroğlu ve arkadaşları ZC706 FPGA platformu üzerinde nicelemeli ağlarla çalışan FINN derin öğrenme çerçevesini sundular. Bizim çalışmamızın da temelini oluşturan FINN çerçevesi kullanılarak CIFAR-10, SVHN ve MNIST veri setleri ile FC ve CNV ağ modelleri için hızlandırıcı uygulamaları yaptılar. Yukarıdaki tabloda çalışmamıza benzerliği açısından geliştirmiş oldukları LFC-Max hızlandırıcısının sonuçlarına yer verilmiştir. FINN ekibi LFC-Max 1 bit nicelemeli model ile MNIST veri setini 1561 KFps'lik bir hızla sınıflandırmayı başardılar. Bizim çalışmamızda aynı veri seti için 1 bit nicelemeli MLP hızlandırıcı ile 951 KFps'lik bir hıza erişilmiştir. Kullandıkları LFC-Max modeli 1024-1024-1024 düğümlü üç tam bağlantı katmanından oluşmakta, 200MHz saat frekansı ile çalışmakta ve 22.6 W enerji tüketmektedir. Bu tezde geliştirilen MLP hızlandırıcı 512-256 düğümlü 2 tam bağlantı katmanından meydana gelmekte ve 100 MHz saat frekansı ile çalışırken sadece 2.05 W enerji tüketmektedir. Ayrıca LFC-Max hızlandırıcı bizim çalışmamızdan yaklaşık 4 kat daha fazla lojik kaynak kullanmıştır. Bu tezde önerilen MLP hızlandırıcının işlem performansı saniyede 1018 milyar işlemidir. FINN tabanlı SFC-Max hızlandırıcı, 8265 GOPs performansı ile benzer çalışmalar arasında en yüksek işlem kabiliyeti olan hızlandırıcıdır ve bizim çalışmamızdan 8 kat daha iyi performansa sahiptir. Bunun nedenlerinden biri kullanmış oldukları FPGA kartının bizim kullanmış olduğumuz ZYNQ XC7Z020 den çok daha yüksek kaynaklara sahip olması iken, bir diğeri de kullandıkları modelde daha çok işlem yapılmasıdır. Güç verimliliği (Watt başına verim) açısından iki çalışma karşılaştırıldığında bizim çalışmamızın %24 daha iyi performans gösterdiği görülmektedir. Umuroğlu ve arkadaşları tarafından yapılan FINN-R isimli çalışma farklı FPGA platformlarında gerçekleştirilmiştir. Bu tezde de kullanılan Pynq Z1 üzerinde geliştirdikleri MLP-4 modeli hızlandırıcı sonuçları karşılaştırma tablosuna alınarak incelenmiştir. Geliştirilen donanımın 974 GOPs işlemi 2.5 W enerji tüketerek gerçekleştirdiği görülmektedir. Bizim geliştirmiş olduğumuz MLP hızlandırıcı ise 1018 GOPs işlemi 2.05 W enerji tüketerek gerçekleştirmektedir. Ayrıca iki donanım kaynak kullanımını açısından değerlendirildiğinde sunmuş olduğumuz hızlandırıcı çok daha az kaynak tüketmektedir. Liang S. ve arkadaşları tarafından yapılan FP-BNN isimli çalışmada FPGA üzerinde bir BNN uygulaması önerdiler. FP-BNN uygulaması 5,9 TOPs gibi yüksek işlem seviyelerine ulaşmıştır. FP-BNN çalışması FINN'dan sonra en yüksek ikinci işlem hızına sahip hızlandırıcıdır. Bu işlem seviyelerine ulaşırken 2210 blok ram ve 182K lojik element kullanmıştır. FP-BNN 26,2 W enerji

tüketirken Watt başına işlem sayısı 225 seviyelerinde hesaplanmıştır. Güç verimliliği açısından bizim çalışmamızla kıyaslandığında, tasarlanmış olduğumuz hızlandırıcı yaklaşık 2,4 kat daha yüksek verime sahiptir. Gao J. ve arkadaşları tarafından geliştirilen FCA-BNN ikilileştirilmiş ağlar için bir FPGA hızlandırıcı donanımdır. FCA-BNN 1,9 TOPs işlem hızına sahip olduğu raporlanmıştır. Bu sonuç 166 MHz saat frekansı ile gerçekleştirilirken, 1.3K BRAM, 152K LUT ve 10 DSP kaynağı kullanmıştır. Bizim uygulamamız ise 100 MHz saat frekansında yaklaşık 1.1 TOPs işlem kapasitesine sahiptir. Bu durum FCA-BNN'in işlem hızında 2 kat daha iyi olmasına karşın bellek kullanımında 15 kat, lojik eleman kullanımında 8 kat daha yüksektir. Güç verimliliği açısından da çalışmamızdan %34 daha düşük performansa sahiptir. Yapmış olduğumuz çalışma incelemiş olduğumuz diğer çalışmalar arasında saniyede yapılan işlem sayısı ve güç verimliliği açısından daha iyi performans sergilemiştir. Genel olarak bakıldığında FINN çerçevesi kullanılarak geliştirilen MLP ve Lenet hızlandırıcılar BNN ağlarında yüksek performans sergilemiştir. Sonraki çalışmalar için bazı önerilerimiz aşağıda verilmiştir.

- Pytorch çerçevesi ile Brevitas kütüphanesi kullanmak modeli küçük parçalara ayırarak ağırlık, giriş ve aktivasyon nicelemelerini ayrı ayrı gerçekleştirmeye olanak sağlar. Bu durum nicelenmiş ağlarda bir hızlandırıcı donanım tasarlamak için önemli bir altyapı oluşturur.
- FINN çerçevesinin sadece BNN ağları ile sınırlı kalmayıp farklı niceleme türlerine kolay adaptasyonu, QNN ağlar için geliştirilen hızlandırıcı çalışmalarında büyük avantaj sağlamaktadır.
- Ayrıca FINN çerçevesi PE/SIMD değerleri ile boru hattını paralelleştirebildiğinden dolayı kaynak kullanımı ve istenilen FPS hızına uygun donanımlar geliştirmeye olanak sağlar.
- FINN ile donanımlar geliştirilirken BRAM, ya da LUTRAM gibi farklı bellek kaynakları seçilebilmektedir. Böylece kullanılacak FPGA platformuna göre kaynak kullanımları değiştirilebilir.
- MLP ağlar MNIST veri setinde yeterli olmasına karşın FashionMNIST ve CIFAR-10, IMAGENet gibi daha kompleks veri setlerinde yetersiz kalmaktadır. Bu tür veri setlerinde Lenet, AlexNet ya da VGG gibi evrimsel ağlar tercih edilmelidir. FINN çerçevesi bu tür standart modellerle çalışabildiği gibi kendi tasarladığımız doğrusal modellere de uygulanabilmektedir.

7. KAYNAKLAR

- [1] E. Cengil and A. Çınar, “a New Approach for Image Classification: Convolutional Neural Network,” *European Journal of Technic EJT*, vol. 6, no. 2, pp. 96–103, 2016.
- [2] I. el Naqa and M. J. Murphy, “What Is Machine Learning?,” in *Machine Learning in Radiation Oncology: Theory and Applications*, I. el Naqa, R. Li, and M. J. Murphy, Eds. Cham: Springer International Publishing, 2015, pp. 3–11. doi: 10.1007/978-3-319-18305-3_1.
- [3] A. Shawahna, S. M. Sait, and A. El-Maleh, “FPGA-Based Accelerators of Deep Learning Networks for Learning and Classification: A Review,” *IEEE Access*, vol. 7, pp. 7823–7859, 2019, doi: 10.1109/ACCESS.2018.2890150.
- [4] S. Rajaraman, S. Candemir, I. Kim, G. Thoma, and S. Antani, “Visualization and interpretation of convolutional neural network predictions in detecting pneumonia in pediatric chest radiographs,” *Applied Sciences*, vol. 8, no. 10, p. 1715, 2018.
- [5] L. Yuan, D. Chen, Y.-L. Chen, N. Codella, X. Dai, J. Gao, H. Hu, X. Huang, B. Li, and C. Li, “Florence: A new foundation model for computer vision,” *arXiv preprint arXiv:2111.11432*, 2021.
- [6] N. Aalami, “Derin öğrenme yöntemlerini kullanarak görüntülerin analizi,” *Eskişehir Türk Dünyası Uygulama ve Araştırma Merkezi Bilişim Dergisi*, vol. 1, no. 1, pp. 17–20, 2020.
- [7] Y. Jiang, X. Zhu, X. Wang, S. Yang, W. Li, H. Wang, P. Fu, and Z. Luo, “R2cnn: rotational region cnn for orientation robust scene text detection,” *arXiv preprint arXiv:1706.09579*, 2017.
- [8] M. Sankaradas, V. Jakkula, S. Cadambi, S. Chakradhar, I. Durdanovic, E. Cosatto, and H. Peter Graf, “A Massively Parallel Coprocessor for Convolutional Neural Networks; A Massively Parallel Coprocessor for Convolutional Neural Networks,” *2009 20th IEEE International Conference on Application-specific Systems, Architectures and Processors*, 2009, doi: 10.1109/ASAP.2009.25.
- [9] S. Cadambi, A. Majumdar Michela, S. Chakradhar, and H. P. Graf, *A Programmable Parallel Accelerator for Learning and Classification*. 2010.
- [10] S. Chakradhar, M. Sankaradas, V. Jakkula, and S. Cadambi, *A Dynamically Configurable Coprocessor for Convolutional Neural Networks*. 2010.

- [11] C. Farabet, B. Martini, B. Corda, P. Akselrod, E. Culurciello, and Y. Lecun, “NeuFlow: A Runtime Reconfigurable Dataflow Processor for Vision,” 2011. doi: 10.1109/CVPRW.2011.5981829.
- [12] M. Peemen, A. A. A. Setio, B. Mesman, and H. Corporaal, *Memory-Centric Accelerator Design for Convolutional Neural Networks*. 2013. doi: 10.1109/ICCD.2013.6657019.
- [13] V. Gokhale, J. Jin, A. Dunder, B. Martini, and E. Culurciello, “A 240 g-ops/s mobile coprocessor for deep neural networks,” in *Proceedings of the IEEE conference on computer vision and pattern recognition workshops*, 2014, pp. 682–687.
- [14] C. Zhang, P. Li, G. Sun, Y. Guan, B. Xiao, and J. Cong, “Optimizing FPGA-based Accelerator Design for Deep Convolutional Neural Networks”, doi: 10.1145/2684746.2689060.
- [15] J. Cong and B. Xiao, “Minimizing computation in convolutional neural networks,” in *International conference on artificial neural networks*, 2014, pp. 281–290.
- [16] K. Ovtcharov, O. Ruwase, J.-Y. Kim, J. Fowers, K. Strauss, and E. S. Chung, “Accelerating deep convolutional neural networks using specialized hardware,” *Microsoft Research Whitepaper*, vol. 2, no. 11, pp. 1–4, 2015.
- [17] C. Zhang, G. Sun, Z. Fang, P. Zhou, P. Pan, and J. Cong, “Caffeine: Toward uniformed representation and acceleration for deep convolutional neural networks,” *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 11, pp. 2072–2085, 2018.
- [18] J. Qiu, J. Wang, S. Yao, K. Guo, B. Li, E. Zhou, J. Yu, T. Tang, N. Xu, and S. Song, “Going deeper with embedded fpga platform for convolutional neural network,” in *Proceedings of the 2016 ACM/SIGDA international symposium on field-programmable gate arrays*, 2016, pp. 26–35.
- [19] Y. Wang, J. Xu, Y. Han, H. Li, and X. Li, “DeepBurning: Automatic generation of FPGA-based learning accelerators for the Neural Network family,” in *2016 53rd ACM/EDAC/IEEE Design Automation Conference (DAC)*, 2016, pp. 1–6. doi: 10.1145/2897937.2898002.
- [20] N. Suda, V. Chandra, G. Dasika, A. Mohanty, Y. Ma, S. Vrudhula, J. Seo, and Y. Cao, “Throughput-optimized OpenCL-based FPGA accelerator for large-scale convolutional neural networks,” in *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2016, pp. 16–25.

- [21] S. I. Venieris and C.-S. Bouganis, “fpgaConvNet: A framework for mapping convolutional neural networks on FPGAs,” in *2016 IEEE 24th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2016, pp. 40–47.
- [22] Z. Liu, Y. Dou, J. Jiang, J. Xu, S. Li, Y. Zhou, and Y. Xu, “Throughput-optimized FPGA accelerator for deep convolutional neural networks,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 10, no. 3, pp. 1–23, 2017.
- [23] Y. Guan, H. Liang, N. Xu, W. Wang, S. Shi, X. Chen, G. Sun, W. Zhang, and J. Cong, “FP-DNN: An Automated Framework for Mapping Deep Neural Networks onto FPGAs with RTL-HLS Hybrid Templates,” 2017, doi: 10.1109/FCCM.2017.25.
- [24] M. Rastegari, V. Ordonez, J. Redmon, and A. Farhadi, “Xnor-net: Imagenet classification using binary convolutional neural networks,” in *European conference on computer vision*, 2016, pp. 525–542.
- [25] M. Kim and P. Smaragdis, “Bitwise neural networks,” *arXiv preprint arXiv:1601.06071*, 2016.
- [26] S. Zhou, Y. Wu, Z. Ni, X. Zhou, H. Wen, and Y. Zou, “Dorefa-net: Training low bitwidth convolutional neural networks with low bitwidth gradients,” *arXiv preprint arXiv:1606.06160*, 2016.
- [27] M. Courbariaux, I. Hubara, D. Soudry, R. El-Yaniv, and Y. Bengio, “Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1,” *arXiv preprint arXiv:1602.02830*, 2016.
- [28] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, “Finn: A framework for fast, scalable binarized neural network inference,” in *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, 2017, pp. 65–74.
- [29] U. Aydonat, S. O’Connell, D. Capalija, A. C. Ling, and G. R. Chiu, “An opencl™ deep learning accelerator on arria 10,” in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 55–64.
- [30] Y. Shen, M. Ferdman, and P. Milder, “Maximizing CNN accelerator efficiency through resource partitioning,” in *2017 ACM/IEEE 44th Annual International Symposium on Computer Architecture (ISCA)*, 2017, pp. 535–547.
- [31] Y. Ma, N. Suda, Y. Cao, S. Vrudhula, and J. Seo, “ALAMO: FPGA acceleration of deep learning algorithms with a modularized RTL compiler,” *Integration*, vol. 62, pp. 14–23, 2018.

- [32] L. Deng, “The MNIST database of handwritten digit images for machine learning research [best of the web],” *IEEE Signal Process Mag*, vol. 29, no. 6, pp. 141–142, 2012.
- [33] W. S. McCulloch and W. Pitts, “A logical calculus of the ideas immanent in nervous activity,” *Bull Math Biophys*, vol. 5, no. 4, pp. 115–133, 1943.
- [34] A. Şeker, B. Diri, and H. H. Balık, “A review about deep learning methods and applications,” *Gazi Mühendislik Bilimleri Dergisi*, vol. 3, no. 3, pp. 47–64, 2017.
- [35] F. Rosenblatt, “The perceptron: a probabilistic model for information storage and organization in the brain.,” *Psychol Rev*, vol. 65, no. 6, p. 386, 1958.
- [36] N. Aalami, “Derin öğrenme yöntemlerini kullanarak görüntülerin analizi,” *Eskişehir Türk Dünyası Uygulama ve Araştırma Merkezi Bilişim Dergisi*, vol. 1, no. 1, pp. 17–20, 2020.
- [37] Y. LeCun, Y. Bengio, and G. Hinton, “Deep learning,” *Nature*, vol. 521, no. 7553, pp. 436–444, 2015, doi: 10.1038/nature14539.
- [38] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” *arXiv preprint arXiv:1506.01497*, 2015.
- [39] Q. Chu, W. Ouyang, H. Li, X. Wang, B. Liu, and N. Yu, “Online multi-object tracking using CNN-based single object tracker with spatial-temporal attention mechanism,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2017, pp. 4836–4845.
- [40] A. Shrivastava, J. Amudha, D. Gupta, and K. Sharma, “Deep learning model for text recognition in images,” in *2019 10Th international conference on computing, communication and networking technologies (ICCCNT)*, 2019, pp. 1–6.
- [41] J. Wang, Y. Yang, J. Mao, Z. Huang, C. Huang, and W. Xu, “Cnn-rnn: A unified framework for multi-label image classification,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2016, pp. 2285–2294.
- [42] K. Zhang, W. Zuo, Y. Chen, D. Meng, and L. Zhang, “Beyond a gaussian denoiser: Residual learning of deep cnn for image denoising,” *IEEE transactions on image processing*, vol. 26, no. 7, pp. 3142–3155, 2017.
- [43] R. Zhang, P. Isola, and A. A. Efros, “Colorful image colorization,” in *European conference on computer vision*, 2016, pp. 649–666.
- [44] I. Goodfellow, Y. Bengio, and A. Courville, *Deep learning*. MIT press, 2016.
- [45] Y. Umuroglu, N. J. Fraser, G. Gambardella, M. Blott, P. Leong, M. Jahre, and K. Vissers, “Finn: A framework for fast, scalable binarized neural network inference,”

- in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 65–74.
- [46] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, and M. Bernstein, “Imagenet large scale visual recognition challenge,” *Int J Comput Vis*, vol. 115, no. 3, pp. 211–252, 2015.
- [47] A. Jain, S. Bhattacharya, M. Masuda, V. Sharma, and Y. Wang, “Efficient execution of quantized deep learning models: A compiler approach,” *arXiv preprint arXiv:2006.10226*, 2020.
- [48] M. Nagel, M. Fournarakis, R. A. Amjad, Y. Bondarenko, M. van Baalen, and T. Blankevoort, “A White Paper on Neural Network Quantization,” Jun. 2021, doi: 10.48550/arxiv.2106.08295.
- [49] M. Kim and P. Smaragdis, “Bitwise neural networks,” *arXiv preprint arXiv:1601.06071*, 2016.
- [50] Y. Bengio, N. Léonard, and A. Courville, “Estimating or propagating gradients through stochastic neurons for conditional computation,” *arXiv preprint arXiv:1308.3432*, 2013.
- [51] M. Blott, T. B. Preußner, N. J. Fraser, G. Gambardella, K. O’Brien, Y. Umuroglu, M. Leeser, and K. Vissers, “FINN-R: An end-to-end deep-learning framework for fast exploration of quantized neural networks,” *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, vol. 11, no. 3, pp. 1–23, 2018.
- [52] E. Wang, J. J. Davis, and P. Y. K. Cheung, “A PYNQ-based framework for rapid CNN prototyping,” in *2018 IEEE 26th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, 2018, p. 223.
- [53] Y. Umuroglu and M. Jahre, “Streamlined deployment for quantized neural networks,” *arXiv preprint arXiv:1709.04060*, 2017.
- [54] S. Liang, S. Yin, L. Liu, W. Luk, and S. Wei, “FP-BNN: Binarized neural network on FPGA,” *Neurocomputing*, vol. 275, pp. 1072–1086, 2018.
- [55] R. Zhao, W. Song, W. Zhang, T. Xing, J.-H. Lin, M. Srivastava, R. Gupta, and Z. Zhang, “Accelerating binarized convolutional neural networks with software-programmable FPGAs,” in *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, 2017, pp. 15–24.
- [56] Z. Nie, Z. Li, L. Wang, S. Guo, Y. Deng, R. Deng, and Q. Dou, “Laius: an energy-efficient FPGA CNN accelerator with the support of a fixed-point training

- framework,” *International Journal of Computational Science and Engineering*, vol. 21, no. 3, pp. 418–428, 2020.
- [57] P. Jokic, S. Emery, and L. Benini, “Binaryeye: A 20 kfps streaming camera system on fpga with real-time on-device image recognition using binary neural networks,” in *2018 IEEE 13th International Symposium on Industrial Embedded Systems (SIES)*, 2018, pp. 1–7.
- [58] Y. Liu, Y. Chen, W. Ye, and Y. Gui, “FPGA-NHAP: A General FPGA-Based Neuromorphic Hardware Acceleration Platform With High Speed and Low Power,” *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 69, no. 6, pp. 2553–2566, 2022.
- [59] J. Gao, Y. Yao, Z. Li, and J. Lai, “Fca-bnn: Flexible and configurable accelerator for binarized neural networks on fpga,” *IEICE Trans Inf Syst*, vol. 104, no. 8, pp. 1367–1377, 2021.
- [60] L. Zhang, X. Tang, X. Hu, Y. Peng, and T. Zhou, “Full-BNN: A Low Storage and Power Consumption Time-Domain Architecture based on FPGA,” in *2022 IEEE 33rd International Conference on Application-specific Systems, Architectures and Processors (ASAP)*, 2022, pp. 98–105.
- [61] Christos Kyrkou, “What are FPGAs?,” <https://hardwarebee.com/the-ultimate-guide-to-fpga-architecture/>, Mar. 04, 2017.
- [62] A. Gholami, S. Kim, Z. Dong, Z. Yao, M. W. Mahoney, and K. Keutzer, “A Survey of Quantization Methods for Efficient Neural Network Inference,” *ArXiv*, vol. abs/2103.13630, 2022.

EKLER

EKLER

EK A: Lenet.py

```
from torch.nn import Module, ModuleList, BatchNorm2d, MaxPool2d, BatchNorm1d
from models.common import CommonWeightQuant, CommonActQuant
from brevitax.nn import QuantIdentity, QuantLinear, QuantConv2d
from models.tensor_norm import TensorNorm
from brevitax.core.restrict_val import RestrictValueType

num_classes=10
weight_bit_width=1
act_bit_width=1
in_bit_width=1
in_channels=1
modelx="Lenet"
device="cpu"

DROPOUT = 0.2

class Lenet(Module):
    def __init__(
        self,
        num_classes,
        weight_bit_width,
        act_bit_width,
        in_bit_width,
        in_channels):

        super(Lenet, self).__init__()

        self.conv_features = ModuleList()
        self.linear_features = ModuleList()

        self.conv_features.append(QuantIdentity( # for Q1.7 input format
            act_quant=CommonActQuant,
            bit_width=in_bit_width,
            min_val=- 1.0,
            max_val=1.0 - 2.0 ** (-7),
            narrow_range=False,
            restrict_scaling_type=RestrictValueType.POWER_OF_TWO))
# *****Conv 1 part*****
        self.conv_features.append(QuantConv2d(
            kernel_size=5,
            in_channels=1,
            out_channels=6,
            bias=False,
            weight_quant=CommonWeightQuant,
            weight_bit_width=weight_bit_width))

        self.conv_features.append(BatchNorm2d(6, eps=1e-4))

        self.conv_features.append(QuantIdentity(
            act_quant=CommonActQuant,
            bit_width=act_bit_width))
# *****Conv 2 part*****
        self.conv_features.append(QuantConv2d(
            kernel_size=5,
            in_channels=6,
            out_channels=16,
            bias=False,
            weight_quant=CommonWeightQuant,
            weight_bit_width=weight_bit_width))

        self.conv_features.append(BatchNorm2d(16, eps=1e-4))

        self.conv_features.append(QuantIdentity(
            act_quant=CommonActQuant,
            bit_width=act_bit_width))
        self.conv_features.append(MaxPool2d(kernel_size=2))
# *****FC 1 part*****
        self.linear_features.append(QuantLinear(
            in_features=256,
            out_features=120,
            bias=False,
            weight_quant=CommonWeightQuant,
```

```

        weight_bit_width=weight_bit_width))

self.linear_features.append(BatchNorm1d(120, eps=1e-4))
self.linear_features.append(QuantIdentity(
    act_quant=CommonActQuant,
    bit_width=act_bit_width))

# *****FC 2 part*****
self.linear_features.append(QuantLinear(
    in_features=120,
    out_features=84,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width))

self.linear_features.append(BatchNorm1d(84, eps=1e-4))
self.linear_features.append(QuantIdentity(
    act_quant=CommonActQuant,
    bit_width=act_bit_width))

# *****FC 3 part*****

self.linear_features.append(QuantLinear(
    in_features=84,
    out_features=10,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width))
self.linear_features.append(TensorNorm())

for m in self.modules():
    if isinstance(m, QuantConv2d) or isinstance(m, QuantLinear):
        torch.nn.init.uniform_(m.weight.data, -1, 1)

def clip_weights(self, min_val, max_val):
    for mod in self.conv_features:
        if isinstance(mod, QuantConv2d):
            mod.weight.data.clamp_(min_val, max_val)
    for mod in self.linear_features:
        if isinstance(mod, QuantLinear):
            mod.weight.data.clamp_(min_val, max_val)

def forward(self, x):
    x = 2.0 * x - torch.tensor([1.0], device=x.device)
    for mod in self.conv_features:
        x = mod(x)
    x = x.view(x.shape[0], -1)
    for mod in self.linear_features:
        x = mod(x)
    return x

```

EK B: MLP.py

```

from torch.nn import Module, ModuleList, BatchNorm2d, MaxPool2d, BatchNorm1d
from models.common import CommonWeightQuant, CommonActQuant
from brevitax.nn import QuantIdentity, QuantLinear, QuantConv2d
from models.tensor_norm import TensorNorm
from brevitax.core.restrict_val import RestrictValueType

num_classes=10
weight_bit_width=1
act_bit_width=1
in_bit_width=1
in_channels=1
modelx="MLP"
device="cpu"

DROPOUT = 0.2

class MLP(Module):
    def __init__(
        self,
        num_classes,

```

```

        weight_bit_width,
        act_bit_width,
        in_bit_width,
        in_channels):

super(Lenet, self).__init__()

self.conv_features = ModuleList()
self.linear_features = ModuleList()

self.conv_features.append(QuantIdentity( # for Q1.7 input format
    act_quant=CommonActQuant,
    bit_width=in_bit_width,
    min_val=- 1.0,
    max_val=1.0 - 2.0 ** (-7),
    narrow_range=False,
    restrict_scaling_type=RestrictValueType.POWER_OF_TWO))
# *****FC 1 part*****
self.linear_features.append(QuantLinear(
    in_features=784,
    out_features=512,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width))

self.linear_features.append(BatchNorm1d(256, eps=1e-4))
self.linear_features.append(QuantIdentity(
    act_quant=CommonActQuant,
    bit_width=act_bit_width))
# *****FC 2 part*****
self.linear_features.append(QuantLinear(
    in_features=512,
    out_features=256,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width))

self.linear_features.append(BatchNorm1d(256, eps=1e-4))
self.linear_features.append(QuantIdentity(
    act_quant=CommonActQuant,
    bit_width=act_bit_width))
# *****FC 3 part*****

self.linear_features.append(QuantLinear(
    in_features=256,
    out_features=10,
    bias=False,
    weight_quant=CommonWeightQuant,
    weight_bit_width=weight_bit_width))
self.linear_features.append(TensorNorm())

for m in self.modules():
    if isinstance(m, QuantConv2d) or isinstance(m, QuantLinear):
        torch.nn.init.uniform_(m.weight.data, -1, 1)

def clip_weights(self, min_val, max_val):
    for mod in self.conv_features:
        if isinstance(mod, QuantConv2d):
            mod.weight.data.clamp_(min_val, max_val)
    for mod in self.linear_features:
        if isinstance(mod, QuantLinear):
            mod.weight.data.clamp_(min_val, max_val)

def forward(self, x):
    x = 2.0 * x - torch.tensor([1.0], device=x.device)
    for mod in self.conv_features:
        x = mod(x)
    x = x.view(x.shape[0], -1)
    for mod in self.linear_features:
        x = mod(x)
    return x

```


EK C: trainer.py

```
net = Lenet(num_classes,
            weight_bit_width,
            act_bit_width,
            in_bit_width,
            in_channels)
use_gpu=False
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(net.parameters(), momentum=0.9, lr=0.02)
nb_epoch=10
print("epoch="+str(nb_epoch))
train_loss_min = np.Inf
train_loss_hist=[]
train_correct=np.zeros(nb_epoch)
train_hist=np.zeros((nb_epoch, 2))
best_epoch_num=0
best_acc_t=.0
t_acc=0
for epoch in range(nb_epoch):
    # monitor training loss
    start = time.time()
    train_loss = 0.0
    tr_acc=0
    t_correct=0
    tr_data_count=0
    net.train()
    for data, target in train_loader:
        tr_data_count+=1
        optimizer.zero_grad()
        train_out = net(data)
        loss_t = criterion(train_out, target)
        loss_t.backward()
        optimizer.step()
        train_loss += loss_t.item() * data.size(0)
        # convert output probabilities to predicted class
        _, pred_t = torch.max(train_out, 1)
        accuracy_t = Accuracy()
        tr_acc+=accuracy_t(pred_t, target)
    tr_acc=(tr_acc/tr_data_count)*100

    train_loss = train_loss / len(train_loader.sampler)

    if (tr_acc)>best_acc_t:
        best_acc_t=tr_acc
        best_epoch_num=epoch

    train_hist[epoch][0]=train_loss
    train_hist[epoch][1]=tr_acc

    end = time.time()
    print('Epoch:{} \tT_Loss: {:.3f}\tT_ACC: %{:0.2f}\t time:{} '.
          format(epoch + 1, train_loss,tr_acc,end-start))

    if train_loss <= train_loss_min:
        #print('Valloss dec ({:0.4f} --> {:0.4f}). Saving model ...'.format(valid_loss_min,
        valid_loss))

    file_name="./export/pt/{_{}_w{}_a{}_batch{}.pt".format(modelx,dset,weight_bit_width,act_bi
    t_width,batch_size)
    torch.save(net.state_dict(), file_name)
    train_loss_min = train_loss

print('\nBest Epoch:{} %{:0.2f}'.format(best_epoch_num+1,best_acc_t))
```

ÖZGEÇMİŞ

Kişisel Bilgiler

Adı Soyadı : Mustafa TAŞCI
Doğum tarihi ve yeri : 21.09.1980- Seydişehir
e-posta : mtasci@bandirma.edu.tr

Öğrenim Bilgileri

Derece	Okul/Program	Yıl
Y. Lisans	Balıkesir Üniversitesi/Elektrik-Elektronik Mühendisliği	2011
Lisans	Balıkesir Üniversitesi/Elektrik-Elektronik Mühendisliği	2015
Lisans	Sakarya Üniversitesi/Elektronik Öğretmenliği	2002
Lise	Seydişehir Endüstri Meslek Lisesi / Elektronik	1997

Yayın Listesi

- [1] İstanbullu A., Taşcı M. (2019). Open source hardware-arduino: Case study on mechanical engineering students design project. International Journal of Engineering Education, 35(5), 1326-1335.
- [2] Demirtaş M., Taşcı M., Kavak S. (2010). PIC16f877 Kullanılarak Kontrol Eğitimi İçin Flyback Dc/Dc Dönüştürücü Tasarımı. Otomatik Kontrol Ulusal Toplantısı 2010