KOCAELİ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ
ANABİLİM DALI

DOKTORA TEZİ

EVRİŞİMSEL SİNİR AĞLARININ FPGA ÜZERİNDE HIZLI VE KAYNAK VERİMLİ KISMİ YAPILANDIRMA TABANLI GERÇEKLENMESİ

HADEE MADADUM

KOCAELİ 2022

KOCAELİ ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ

ANABİLİM DALI

DOKTORA TEZİ

EVRİŞİMSEL SİNİR AĞLARININ FPGA ÜZERİNDE HIZLI VE KAYNAK VERİMLİ KISMİ YAPILANDIRMA TABANLI GERÇEKLENMESİ

HADEE MADADUM

Prof. Dr. Yaşar Danışman,	**BECERİKLİ Kocaeli Üniversitesi**	**.....**
Prof. Dr. Cabir Jüri Üyesi,	**VURAL Marmara Üniversitesi**	**.....**
Prof. Dr. Cemil Jüri Üyesi,	**Öz Sakarya Üniversitesi**	**.....**
Doç. Dr. Orhan Jüri Üyesi,	**AKBULUT Kocaeli Üniversitesi**	**.....**
Doç. Dr. Suhap Jüri Üyesi,	**ŞAHİN Kocaeli Üniversitesi**	**.....**

Tezin Savunulduğu Tarih: 20.06.2022

ETİK BEYAN VE ARAŞTIRMA FONU DESTEĞİ

Kocaeli Üniversitesi Fen Bilimleri Enstitüsü tez yazım kurallarına uygun olarak hazırladığım bu tez/proje çalışmasında,

- Bu tezin bana ait, özgün bir çalışma olduğunu,
- Çalışmamın hazırlık, veri toplama, analiz ve bilgilerin sunumu olmak üzere tüm aşamalarında bilimsel etik ilke ve kurallara uygun davrandığımı,
- Bu çalışma kapsamında elde edilen tüm veri ve bilgiler için kaynak gösterdiğimi ve bu kaynaklara kaynakçada yer verdiğimi,
- Bu çalışmanın Kocaeli Üniversitesi'nin abone olduğu intihal yazılım programı kullanılarak Fen Bilimleri Enstitüsü'nün belirlemiş olduğu ölçütlere uygun olduğunu,
- Kullanılan verilerde herhangi bir tahrifat yapmadığımı,
- Tezin herhangi bir bölümünü bu üniversite veya başka bir üniversitede başka bir tez çalışması olarak sunmadığımı,

beyan ederim.

☒ Bu tez/proje çalışmasının herhangi bir aşaması hiçbir kurum/kuruluş tarafından maddi/alt yapı desteği ile desteklenmemiştir.

☐ Bu tez/proje çalışması kapsamında üretilen veri ve bilgiler …… tarafından ……. no'lu proje kapsamında maddi/alt yapı desteği alınarak gerçekleştirilmiştir.

Herhangi bir zamanda, çalışmamla ilgili yaptığım bu beyana aykırı bir durumun saptanması durumunda, ortaya çıkacak tüm ahlaki ve hukuki sonuçları kabul ettiğimi bildiririm.

Hadee MADADUM

YAYIMLAMA VE FİKRİ MÜLKİYET HAKLARI

Fen Bilimleri Enstitüsü tarafından onaylanan lisansüstü tezimin tamamını veya herhangi bir kısmını, basılı ve elektronik formatta arşivleme ve aşağıda belirtilen koşullarla kullanıma açma izninin Kocaeli Üniversitesi'ne verdiğimi beyan ederim. Bu izinle Üniversiteye verilen kullanım hakları dışındaki tüm fikri mülkiyet haklarım bende kalacak, tezimin/projemin tamamının ya da bir bölümünün gelecekteki çalışmalarda (makale, kitap, lisans ve patent vb.) kullanımı bana ait olacaktır.
Tezin kendi özgün çalışmam olduğunu, başkalarının haklarını ihlal etmediğimi ve tezimin tek yetkili sahibi olduğumu beyan ve taahhüt ederim. Tezimde yer alan telif hakkı bulunan ve sahiplerinden yazılı izin alınarak kullanılması zorunlu metinlerin yazılı izin alarak kullandığımı ve istenildiğinde suretlerini Üniversiteye teslim etmeyi taahhüt ederim.
Yükseköğretim kurulu tarafından yayınlanan ***"Lisansüstü Tezlerin Elektronik Ortamda Toplanması, Düzenlenmesi ve Erişime Açılmasına İlişkin Yönerge"*** kapsamında tezim aşağıda belirtilen koşullar haricinde YÖK Ulusal Tez Merkezi/ Kocaeli Üniversitesi Kütüphaneleri Açık Erişim Sisteminde erişime açılır.

☐ Enstitü yönetim kurulu kararı ile tezimin/projemin erişime açılması mezuniyet tarihinden itibaren 2 yıl ertelenmiştir.

☐ Enstitü yönetim kurulu gerekçeli kararı ile tezimin/projemin erişime açılması mezuniyet tarihinden itibaren 6 ay ertelenmiştir.

☒ Tezim/projem ile ilgili gizlilik kararı verilmemiştir.

Hadee MADADUM

ÖNSÖZ VE TEŞEKKÜR

İlk olarak, bu tez çalışmasını tamamlamam için bana güç ve sağlık veren yüce Allah'a şükürler olsun. Ayrıca tez süresi boyunca benden desteklerini esirgemeyen herkese teşekkürlerimi sunmayı bir borç bilirim.

Bunlar içerisinde, değerli fikirleriyle beni sürekli destekleyen danışmanım ve hocam, Prof. Dr. Yaşar BECERİKLİ ’ye minnettarım. Samimiyeti ve motivasyonu beni derinden etkilenmiştir. Bana sundukları için son derece teşekkür ederim.

Tez izleme jürimde yer alan Prof. Dr. Cabir VURAL ve Doç. Dr. Orhan AKBULUT’a bilgi, fikir ve önerileri için saygılarımı sunarım.

Bu tez, Yurtdışı Türkler ve Akraba Topluluklar Başkanlığı (YTB) tarafından verilen doktora destek bursu ile tamamlanmıştır. Bu destekler olmadan bu çalışmanın nihayete ermesi mümkün değildi. Bu nedenle bu kuruma teşekkürü bir borç bilirim.

Bu tezin Türkçe ‘ye çevrilmesinde desteklerini esirgemeyen Dr. Öğr. Üyesi Burcu KIR SAVAŞ ve Harun Reşit KIRBIYIK'a teşekkürü bir borç bilirim.

Burada yine zikretmeden geçemeyeceğim ailem ve akrabalarıma sevgi, dua, sabır, özen ve fedakarlıklardan dolayı sevgilerimi sunarım. Bunlarla beraber, sürekli teşviki için arkadaşlara ve araştırmacı meslektaşlarıma teşekkür ederim.

Son olarak, bu tez çalışmasının, FPGA ve ConNN hakkında bilgi sahibi olmak isteyenlere ve konuya ilgi duyan akademisyenlere faydalı olmasını diliyorum.

Haziran – 2022 Hadee MADADUM

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

x	: Girdi
y	: Çıktı
w	: Ağırlık
S	: Adım
T_m	: M boyutunda döşeme parametresi
T_n	: N boyutunda döşeme parametresi
T_k	: K boyutunda döşeme parametresi
U_m	: M boyutunda açma parametresi
U_n	: N boyutunda açma parametresi
U_k	: K boyutunda açma parametresi
T_{pr}	: Toplam kısmi yeniden konfigürasyon Süresi
PB_{size}	: Kısmi bit akışı dosyası boyutu
Δ	: Nicemleme adım boyutu
Q_{log}	: Logaritmik nicemleme
Q_{slog}	: Qset ile logaritmik nicemleme
P_i	: Yüzdelik dilim

Kısaltmalar

ConNN	: Convolutional Neural Network (Evrişimsel Sinir Ağları)
FPGA	: Field Programmable Gate Array (Alan Programlanabilir Kapı Dizileri)
CPU	: Central Processing Unit (Merkezi İşlem Birimi)
GPU	: Graphics Processing Unit (Grafik İşlem Birimi)
ASIC	: Application-Specific Integrated Circuits (Uygulamaya Özel Tümleşik Devre)
MAC	: Multiply-Accumulate unit (Çarpma-Biriktirme İşlemi)
FMs	: Feature Maps (Özellik Haritaları)
CONV	: Convolutional layer (Evrişimsel katman)
FC	: Fully Connected layer (Tam bağlantı katmanı)
ReLU	: Rectified Linear Unit (Doğrultulmuş Lineer Birim)
FLOPs	: Floating Point Operations (Kayan Nokta İşlemleri)
OP	: Operation (İşlem)
CLB	: Configurable Logic Block (Yapılandırılabilir Mantık Bloğu)
DSP	: Digital Signal Processing (Sayısal İşaret İşleme)
LUT	: Lookup Table (Arama Tablosu)
BRAM	: Block RAM (RAM bloğu)
FF	: Flip-flop
HDL	: Hardware Description Language (Donanım Tanımlama Dili)
PE	: Processing Element (İşleme Elemanı)
GEMM	: General Matrix Multiplications (Genel Matris Çarpımları)
Buff	: Buffer (Arabellek)
PS	: Processing System (İşleme Sistemi)
PL	: Programmable Logic (Programlanabilir Mantık)
GOP/s	: Giga-operation Per Second (Saniyede Giga İşlemi)
SP	: Static Partition (Statik Bölüm)

RP : Reconfigurable Partition (Yeniden Yapılandırılabilir Bölüm)
PR : Partial Reconfiguration (Kısmi Yeniden Konfigürasyon)
RM : Reconfigurable Module (Yeniden Yapılandırılabilir Modül)
BU : Buffer Underutilization (Arabellek Eksik Kullanımı)
IL : Integer Bit Length (Tamsayı Bit Uzunluğu)
FL : Fraction Bit Length (Kesir Bit Uzunluğu)
FSR : Full Scale Range (Tam Ölçek Aralığı)
mAP : Mean Average Precision (Genel Ortalama Kesinlik)
ConNN : Convolutional Neural Network (Konvolüsyonel Sinir Ağları)
FPGA : Field Programmable Gate Array (Alan Programlanabilir Kapı Dizileri)
CPU : Central Processing Unit (Merkezi İşlem Birimi)
GPU : Graphics Processing Unit (Grafik İşlem Birimi)
ASIC : Application-Specific Integrated Circuits (Uygulamaya Özel Tümleşik Devre)
MAC : Multiply-Accumulate unit (Çarpma-Biriktirme İşlemi)
FMs : Feature Maps (Özellik Haritaları)
CONV : Convolutional layer (Konvolüsyonel katman)
FC : Fully Connected layer (Tam bağlantı katmanı)
ReLU : Rectified Linear Unit (Doğrultulmuş Lineer Birim)
FLOPs : Floating Point Operations (Kayan Nokta İşlemleri)
OP : Operation (İşlem)
CLB : Configurable Logic Block (Yapılandırılabilir Mantık Bloğu)
DSP : Digital Signal Processing (Sayısal İşaret İşleme)
LUT : Lookup Table (Arama Tablosu)
BRAM : Block RAM (RAM bloğu)
FF : Flip-flop
HDL : Hardware Description Language (Donanım Tanımlama Dili)
PE : Processing Element (İşleme Elemanı)
GEMM : General Matrix Multiplications (Genel Matris Çarpımları)
Buff : Buffer (Arabellek)
PS : Processing System (İşleme Sistemi)
PL : Programmable Logic (Programlanabilir Mantık)
GOP/s : Giga-operation Per Second (Saniyede Giga İşlemi)
SP : Static Partition (Statik Bölüm)
RP : Reconfigurable Partition (Yeniden Yapılandırılabilir Bölüm)
PR : Partial Reconfiguration (Kısmi Yeniden Konfigürasyon)
RM : Reconfigurable Module (Yeniden Yapılandırılabilir Modül)
BU : Buffer Underutilization (Arabellek Eksik Kullanımı)
IL : Integer Bit Length (Tamsayı Bit Uzunluğu)
FL : Fraction Bit Length (Kesir Bit Uzunluğu)
FSR : Full Scale Range (Tam Ölçek Aralığı)
mAP : Mean Average Precision (Genel Ortalama Kesinlik)

EVRİŞİMSEL SİNİR AĞLARININ FPGA ÜZERİNDE HIZLI VE KAYNAK VERİMLİ KISMİ YAPILANDIRMA TABANLI GERÇEKLENMESİ

ÖZET

Evrişimsel Sinir Ağları (ConNN/CNN) algoritması, klasik makine öğrenmesi yöntemlerine kıyasla ConNN modelleri doğrulukta önemli bir gelişme sağlaması sayesinde, özellikle bilgisayarla görme alanında bilişsel uygulamalar için verimli algoritmalardan biridir. Ancak ConNN algoritması, yüksek performanslı bilgi işlem cihazları gerektiren bilgi işlem ve bellek açısından yoğun bir görevdir. Buna göre, yüksek performans ve enerji verimliliğine sahip bir donanım işlemcisini belirlemek için ConNN modelinin çeşitli platformlarda konuşlandırılması incelenmiştir.

Alanda Programlanabilir Kapı Dizisi (FPGA) platformları, bilgi işlem yeteneği, güç verimliliği ve kullanıcı dostu arayüzler açısından FPGA cihazlarındaki son gelişmeler, ConNN modellerini hızlandırmak için güçlü bilgi işlem platformlarından biri olarak ortaya çıkmıştır. FPGA cihazının avantajları, paralel hesaplama yeteneği ve mantık modüllerinin yeniden programlanabilmesidir. Ancak ConNN modelinin FPGA üzerinde uygulanması kaynak kısıtlamaları ile sınırlıdır. Bu nedenle, kaynak kısıtlamaları altında FPGA'nın optimum performansını elde etmek için ConNN'nin kaynak verimliliği ile hızlandırılması araştırılmıştır.

Bu tezde, kaynak kısıtlı bir ortamda ConNN modelini hızlandırmak için FPGA tabanlı bir donanım hızlandırıcı önerilmiştir. Önerilen hızlandırıcı, verimli donanım optimizasyon yöntemleri ve donanım davranışının sayısal analizi kullanılarak tasarlanmıştır. Performans değerlendirmesi için AlexNet, ResNet-18 ve YOLOv3-tiny gibi farklı ConNN modelleri uygulanmıştır. Ayrıca, dinamik donanım tasarımı kullanarak kaynak verimliliğini daha da artırmak için uyarlanabilir bir hızlandırıcı önerilmiştir. Uyarlanabilir hızlandırıcının mimarisi, FPGA'nin kısmi yeniden yapılandırma yeteneği kullanılarak bilgi işlem ve depolama gereksinimlerine göre çalışma zamanında yeniden yapılandırılır. Önerilen uyarlamalı tasarım, statik tabanlı hızlandırıcı tarafından geliştirilemeyen kaynak yetersiz kullanımını iyileştirmiştir.

Ayrıca, yüksek çözünürlüklü logaritmik nicemleme yöntemini kullanan bir ConNN sıkıştırma tekniği önerilmiştir. Önerilen yöntemi kullanarak, nicemlenmiş ConNN modelleri, yeniden eğitim süreci olmaksızın orijinal doğruluğa yakın bir doğruluk oranı elde etmiştir. Ek olarak, düşük bit genişlikli nicemleme yönteminin performansını geliştirmek için bir aykırı ağırlık nicemleme yöntemi gösterilmiştir. Önerilen yöntem, büyük bir ağırlığı iki küçük değerle değiştirmektedir. Böylece logaritmik nicemleme, düşük bit genişliği kullanılarak rahatlıkla gerçekleştirilebilir. Nicemlenmiş ConNN modellerini gerçekleştirmek için bit kaydırma tabanlı bir hızlandırıcı da sunulmuştur. Deneysel sonuçlar, önerilen hızlandırıcının çarpma tabanlı tasarıma kıyasla önemli ölçüde daha iyi kaynak verimliliği sağladığını göstermiştir.

Anahtar Kelimeler: Evrişimsel Sinir Ağları, FPGA, Görüntü Sınıflandırma, Nesne Algılama, Veri Nicemleme.

FAST AND RESOURCE EFFICIENT IMPLEMENTATION OF CONVOLUTIONAL NEURAL NETWORKS ON FPGA BASED ON PARTIAL RECONFIGURATION

ABSTRACT

Convolutional Neural Networks (ConNN/CNN) algorithm is one of the efficient algorithms for cognitive applications, especially in the field of computer vision, as ConNN models provide a significant improvement in accuracy compared to classical machine learning methods. However, ConNN algorithm is a computing and memory-intensive task that requires a high-performance computing device. Accordingly, the deployment of ConNN model has been studied on various platforms to obtain a hardware processor with high performance and energy efficiency.

Field Programmable Gate Array (FPGA) has emerged as one of the powerful computing platforms to accelerate ConNN model due to recent improvements in FPGA devices with respect to computing capability, power efficiency and user-friendly development tools. The advantages of the FPGA device are parallel computing capability and re-programmability of the logic modules. However, the implementation of ConNN model on FPGA is limited by resource constraints. For this reason, accelerating ConNN with resource efficiency has been studied to achieve optimum performance of FPGA devices under resource constraints.

In this thesis, an FPGA-based hardware accelerator is proposed to perform ConNN model in a resource-constrained environment. The proposed accelerator is designed using hardware optimization techniques and numerical analysis of hardware behaviour. Different ConNN models including AlexNet, ResNet-18 and YOLOv3-tiny are implemented for performance evaluation. Moreover, an adaptive hardware accelerator is demonstrated to further increase resource efficiency using dynamic hardware design. The architecture of the adaptive accelerator is able to be reconfigured at runtime according to the computing and storage requirements using partial reconfiguration capability. The proposed adaptive design solves resource underutilization that cannot be improved by the static-based accelerator.

Additionally, in the thesis, a ConNN compression technique using high-resolution logarithmic quantization is proposed. Applying the proposed method, quantized ConNN models achieve accuracy close to original accuracy without the retraining process. Besides this, an outlier weights separation method is demonstrated to improve the performance of low bit-width quantization method. The proposed method replaces a large weight with two smaller values. Thus, logarithmic quantization can be conveniently performed using a low bit-width. A bit-shift-based accelerator is also presented to perform quantized ConNN models. Experimental results showed that the proposed accelerator provides significantly better resource efficiency compared to the multiply based design.

Keywords: Convolutional Neural Networks, FPGA, Image Classification, Object Detection, Data Quantization.

1. GİRİŞ

Son yıllarda, Evrişimsel Sinir Ağları (ConNN)[1] , görüntü sınıflandırma, nesne algılama ve sinirsel dil işleme gibi bilişsel görevlerde çok yüksek doğrulukta sonuçlar vermiştir. Örneğin, el yazısı rakam tanıma (Lecun ve diğ., 1998; Madadum ve diğ., 2022; Maitra ve diğ., 2015) yüz algılama ve tanıma (Jiang ve diğ., 2017; H. Li ve diğ., 2015) konuşma tanıma (Abdel-Hamid ve diğ., 2014; Amodei ve diğ., 2016) görüntü sınıflandırma (Krizhevsky ve diğ., 2012; Qin ve diğ., 2020; Simonyan ve diğ., 2014; Szegedy ve diğ., 2017), nesne algılama (W. Liu ve diğ., 2016; Redmon ve diğ., 2018) çalışmaları verilebilir.

Sayısal bilginin yükselişi ile ConNN daha da zor problemleri çözmek için çeşitli alanlarda sürekli olarak kullanılmıştır. Bu nedenle, beklenen doğruluğu elde etmek için ConNN mimarisi daha karmaşık olma eğilimindedir. ConNN yapılarında artan bu eğilimin başlangıç noktası, 2012 yılında AlexNet (Krizhevsky ve diğ., 2012) adlı bir ConNN modelinin bilgisayarla görme zorluğu olan ILSVRC yarışmasında geleneksel makine öğrenimi yöntemlerinden daha iyi performans göstermesiydi.

AlexNet 61 milyon parametreye sahiptir ve ImageNet veri kümesinde (Russakovsky ve diğ., 2015) %84,6 ilk 5 doğruluk oranı elde edebilir. Son zamanlarda öne çıkan ConNN modeli FixEfficientNet-L2'dir. EfficientNet (Tan ve diğ., 2019) adlı bu ConNN modelinin performansını artırmak için Facebook AI araştırma grubu (Touvron ve diğ., 2020) tarafından geliştirilmiştir. FixEfficientNet-L2, 480 milyon parametreye sahiptir ve ImageNet veri kümesinde %98,7'lik en iyi 5 doğruluk oranı sağlayabilmektedir. Daha derin tasarım ve daha karmaşık ConNN modelleri doğruluk performansını geliştirebilse de bilgi işlemsel işlemlerinin sayısı ve depolama gereksinimleri de büyük ölçüde artmıştır. Bu tür bilgi işlem ve yoğun bellek gerektiren görevleri gerçekleştirmek için

[1] Bu tezde Evrişimsel Sinir Ağları (Convolutional Neural Networks) kısaltılması ConNN olarak verilmiştir. CNN uzun süredir Hücresel Sinir Ağları (Cellular Neural Networks) tanımının kısaltılması olarak ve CoNN uzun süredir İşbirlikçi Sinir Ağları (Cooperative Neural Networks) sinir ağlarının kısaltılması olarak literatürde yer almaktadır.

ConNN hızlandırmaya yönelik güçlü ve verimli donanım platformlarının keşfi üzerinde çalışılmaktadır.

Merkezi İşlem Birimi (CPU), bilgisayar ve mikro denetleyici gibi cihazlarda bulunabilen bir işlemci bileşenidir. CPU, neredeyse her türlü bilgi işlem işini gerçekleştirmek için tasarlanmıştır. Son CPU'lar, çoklu bilgi işlem çekirdeği ve çoklu kullanım yetenekleri ile performanslarını geliştirmektedir. Ancak CPU, paralel bilgi işlem eksikliği nedeniyle ConNN hesaplamasında problemler vardır. Sonuç olarak, düşük işlem oranı performansı (Nurvitadhi ve diğ., 2016) ve düşük enerji verimliliği (Qasaimeh ve diğ., 2019) vermiştir.

İstenilen performansı elde etmek için, çoğu ConNN uygulama çerçevesi aracı tarafından Grafik İşlem Birimi (GPU) kullanılmıştır. GPU, tipik olarak tek bir görüntüyü işlemek için oluşturulur ve gecikmenin kritik olduğu bilgisayar grafikleri ve görüntü işleme uygulamaları için kullanılır. GPU, yüksek bellek bant genişliği ve işlem oranı için yüksek düzeyde paralel yapı tasarımına sahip büyük veri bloklarından ve işlemcilerden oluşur. Bu nedenle hem eğitim hem de iletme süreçlerinde (Vasudevan ve diğ., 2017) ConNN bilgi işlemesinin ana işlemleri olan kayan-nokta ve matris tabanlı işlemler için çok uygundur. Ancak, ConNN uygulaması için GPU kullanmanın ana dezavantajı enerji tüketimidir (Nurvitadhi ve diğ., 2017). Bu nedenle, GPU tabanlı ConNN uygulamalarının çoğu, yeterli güç kaynağına sahip bir bulut servisi olarak büyük iş istasyonlarına veya sunuculara dağıtılmaktadır.

Uygulamaya Özel Tümleşik Devre (ASIC), hem güçlü performans hem de yüksek enerji verimliliği sağlayan bir platformdur. Bu platform, bellek ve bilgi işlem yapısında tamamen özelleştirilebilmektedir. Bu nedenle hem işlem oranı hem de enerji verimliliğini büyük ölçüde artırabilmektedir (Moolchandani ve diğ., 2021). Bununla birlikte, ASIC' in yeniden konfigürasyonu zayıftır ve bu da ilk uygulama yüksek maliyetlerine neden olur. Ayrıca ASIC üzerinde devre tasarlamak, derin donanım bilgisi gerektirir ve geliştirme döngüsü çok karmaşıktır. Bu nedenle, ASIC tabanlı hızlandırıcılar henüz ConNN hızlandırması için birincil platform haline gelmemiştir.

Son birkaç yılda, Alan Programlanabilir Kapı Dizileri (FPGA) platformları, performans iyileştirmeleri gösterdikleri ve kullanıcı dostu geliştirme araçları sağladıkları için ConNN hızlandırıcıları adına giderek daha çekici bir donanım haline gelmiştir. FPGA performans

olarak, GPU ve ASIC'in arasındadır. FPGA, GPU'dan 2,3 - 4,3× daha yüksek güç verimliliği sağlarken ASIC, FPGA'ya kıyasla güç verimliliğini 10× artırabilmektedir. FPGA, GPU'ya kıyasla 2,1 - 3,5× daha yüksek aktarım hızı performansı sunar. Ancak FPGA, ASIC'den 6,3× daha düşük performans gerçekleştirmektedir (Boutros ve diğ., 2018). ASIC'lerin FPGA'lardan daha iyi performans göstermesine rağmen, FPGA'ların uygulanması ASIC'lerden daha kolay ve pratiktir. Yeniden yapılandırma yeteneği, donanım düzeyinde son derece esnek tasarım sağlamıştır. Tasarımcılar farklı veri türleriyle veri yapılarını özelleştirebilmektedir. Ayrıca, FPGA satıcıları, kullanıcıların C veya C++ programlama dilini kullanarak FPGA üzerinde donanım uygulayabilecekleri yüksek seviyeli sentez (HLS) programı gibi geliştirme araçları sağlamıştır. Bu, daha düşük tasarım maliyetleriyle ve hızlı geliştirme ile sonuçlanmıştır. ConNN bilgi işlem için donanım geliştirme platformları ve özellikleri Şekil 1.1'de özetlenmiştir.

Şekil 1.1. ConNN için ortaya çıkan donanım platformları

Benzer şekilde, diğer platformlarda olduğu gibi, ConNN'nin FPGA üzerinde uygulanmasında da zorluklarla karşılaşılmıştır. İlk olarak, FPGA, bellek kısıtlaması sunar. Ancak ConNN algoritması, bellek yoğun kullanan bir görevdir. Örneğin, AlexNet modeli yaklaşık 61 milyon parametreye sahiptir ve tüm katmanlar (Siu ve diğ., 2018) için ve ağırlıkları depolamak için yaklaşık 150 MB bellek kapasitesi gerekir. Ancak FPGA, hatta yaklaşık 11.2 ila 45 MB çip üstü belleğe sahip olan küçük ölçekli FPGA bile, bu tür kapasite gereksinimleri için çip üstü bellek sağlayamaz. İkinci zorluk, bilgi işlem modüllerinin maliyetidir. ConNN, milyarlarca işlemle başa çıkmak için yüksek paralellik gerektirir. Ancak FPGA, paralel bilgi hesaplamasını mümkün kılmak için sınırlı bilgi işlem kaynakları sağlar. Bu nedenle, beklenen performansı sağlamak için mevcut kaynakları yönetmek zorlu bir iştir. Ayrıca, FPGA, kullanıcıların otomatik araçlarla sayısal donanım kurmasına olanak tanıyan programlanabilir mantık sağlamıştır. Ancak,

uygun donanım tasarımı bilgisi olmadan, nicemleme yöntemi veya veri akışı optimizasyonu gibi tekniklerin uygulanması tasarımcılar için zorluklardan biri olarak gösterilebilir.

Bu tezde, küçük ölçekli FPGA'yı hedefleyen ConNN hızlandırma uygulaması sunulmaktadır. Bunu başarmak için verimli donanım tasarımı stratejileri, bilgi işleme optimizasyon teknikleri ve ConNN nicemleme yöntemleri araştırılmıştır. Bu tezin sundukları, aşağıdakileri içermektedir:

- Küçük ölçekli FPGA üzerinde kaynak açısından verimli bir ConNN hızlandırıcı tanıtılmıştır. Kaynak kısıtlamaları altında bilgi işlemsel gecikmeyi ve bellek erişim ek yükünü en aza indirmeyi amaçlayan bir evrişim optimizasyonu analizi sunulmaktadır. Önerilen yöntem, görüntü sınıflandırma ve nesne algılama için yaygın olarak kullanılan ConNN modellerinde değerlendirilmiştir.
- Esnek bir bilgi işlem birimi kullanan ConNN çıkarımı için uyarlanabilir bir hızlandırıcı önerilmiştir. Bu hızlandırıcı, kaynak yetersiz kullanım sorununu çözmek için hesaplama ve depolama taleplerine göre yeniden yapılandırılabilen dinamik bir donanım yapısı sağlar. ConNN modelini hızlandırıcının donanımıyla eşleştirmek için, katmanları yapısal benzerliklerine göre kümeler halinde gruplamak için bir ağ bölümü tanıtılmıştır.
- ConNN modelinin kaynak gereksinimlerini azaltmak için logaritmik tabanlı nicemleme kullanan bir veri sıkıştırma tekniği sağlanmıştır. Basit bir logaritmik kuantizasyonun performansını geliştirmek için yüksek çözünürlüklü logaritmik ölçek ve aykırı değer ağırlıkları azaltma yöntemleri önerilmiştir. Ayrıca hızlandırıcıdaki çarpanın bit kaydırıcı ile değiştirildiği, FPGA cihazları için donanım dostu bir tasarım sağlayan çarpansız bir hızlandırıcı sunulmuştur.

Bu tezin geri kalanı şu şekilde organize edilmiştir. Bölüm 2, FPGA üzerinde derin öğrenme ile nesne sınıflaması, ConNN'nin tanıtımı, FPGA üzerinde ConNN'nin hızlandırıcı, tezin motivasyonunu ve literatürün bir incelemesini sunmaktadır. Bölüm 3’te, evrişim hızlandırma teknikleri ve ConNN hızlandırıcı için donanım tasarım stratejileri verilmektedir. Bölüm 4’de, kısmi yeniden yapılandırma kullanan uyarlanabilir bir hızlandırıcı sunmaktadır. Bölüm 5, hızlandırıcı için logaritmik tabanlı nicemleme ve

donanım dostu mimariyi sunmaktadır. Bölüm 6, logaritmik nicemleme geliştirmek için aykırı değer ağırlıkları ayrımı adı verilen bir yöntemi açıklamaktadır. Bölüm 7, ConNN modelini oluşturmak için FPGA kullanımını ve çalışmalarımızda kullanılan geliştirme araçlarını göstermektedir.

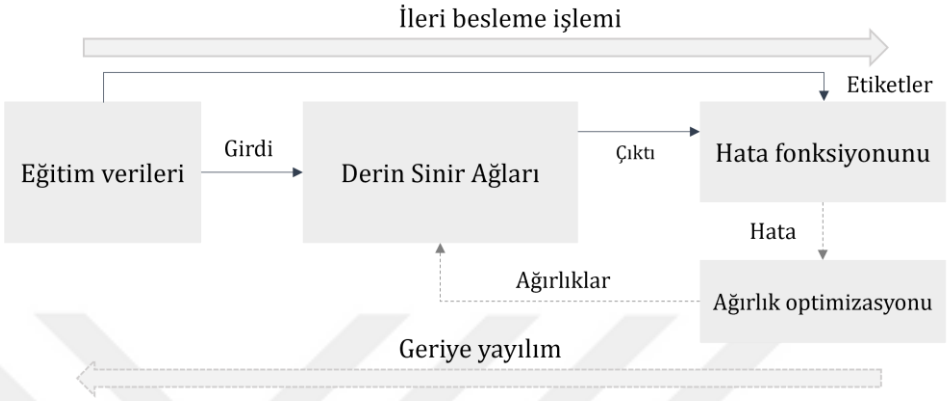
2. FPGA ÜZERİNDE DERİN ÖĞRENME İLE NESNE SINIFLAMASI

Bu bölümde ilk olarak derin öğrenme ve derin öğrenmede kullanılan yöntemler tanıtılmaktadır. Konvolüsyonel sinir ağları adı verilen bir derin öğrenme alt kümesi sunulmaktadır. Bu girişten sonra, bir nesne sınıflandırması ve nesne sınıflandırması için bazı etkili yöntemler gösterilmektedir. Daha sonra, FPGA üzerinde bir ConNN hızlandırması sunulur. Son olarak, motivasyonumuz ve ConNN'nin hızlandırılması ile ilgili önceki çalışmalar anlatılmaktadır.

Derin öğrenme, sinir ağlarına dayalı olarak yapılandırılmış makine öğrenmesi algoritmalarından biridir. Sınıflandırma ve regresyon gibi yapay zekâ görevlerini çözmek için yaygın olarak kullanılır. Sinir ağları, bir dizi nöron ve bağlantıdan oluşur. Her nöron birçok girdi ve bir çıktıdan meydana gelir. Bu yöntem, çıktıyı sunmak için aktivasyon adı verilen doğrusal olmayan bir işlev tarafından takip edilen ağırlıklı girdilerin toplamını hesaplayan bir algılayıcı algoritması kavramını benimser. Nöron gruplarına katman denmektedir. Yapay sinir ağlarında girdi, çıktı ve gizli katmanları bulunur. Girdi katmanı, önceki nöronlarla bağlantısı olmayan nöronlardan oluşmaktadır. Çıkış katmanı, ardılı olmayan nöronlardan oluşmaktadır. Gizli katman, girdi ve çıktı katmanı arasındaki katmandır. Gizli katman sayısı fazla ise bu, derin sinir ağı olarak adlandırılır. Nöronlar sadece farklı katmanlardaki nöronlara bağlıdır. Nöronlar arasındaki her bağlantının, ileri nöronun girdi sinyalini modüle etmek için ve önceki nöronun çıktısıyla çarpmak için kullanılan bir ağırlığı vardır. Nöron i'nin çıktısı, ağırlık w_{ij} ile nöron j'nin girdisine bağlanmaktadır. Ağırlık, öğrenme sürecinde ayarlanabilen, öğrenilebilir bir parametredir. Bu süreç eğitim olarak da bilinmektedir.

Derin sinir ağlarının eğitimi bir optimizasyon problemi olarak kabul edilmiştir. Gradyan azalma, Newton yöntemi, Conjugated gradyan ve Levenberg-Marquardt algoritması gibi eğitim için birkaç verimli optimizasyon algoritması kullanılabilmektedir. En yaygın olarak kullanılan teknik, hata fonksiyonunun gradyanını hesaplayan ve bu gradyanı ağırlık parametresini ayarlamak için kullanan birinci dereceden bir optimizasyon yöntemi olan bayır iniş tekniğidir (stochastic gradyan, mini-batch, vb.). Ağırlığı güncellemeden önce, gradyan değeri öğrenme oranı adı verilen bir parametre ile çarpılır. Bu değer sabit bir değer olabileceği gibi eğitim sırasında güncellenebilen dinamik bir değer de

olabilmektedir. Eğitim süreci için geri yayılım algoritması benimsenmiştir. Geri yayılımlı gradyanı hesaplamak için ağın, girdiden çıktı nöronuna bir ileri besleme gerçekleştirerek ağırlığın çıktı nöronuna etkisini bilmesi gerekir. Bu süreç çıkarım olarak da bilinmektedir. Şekil 2.1 Derin sinir ağları kavramını, çıkarım ve eğitim sürecini göstermektedir.

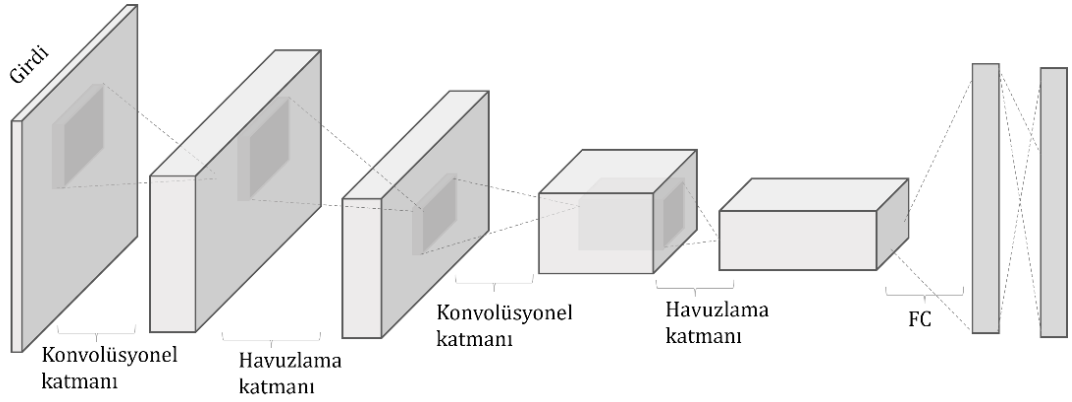


Şekil 2.1. Derin sinir ağları, çıkarım ve eğitim sürecine genel bakış

Ağın çıktısı, regresyon görevlerinde sürekli ve vektör de olabilir. Sınıflandırma problemlerinde ağır çıktısı bir vektör olup, vektörün bileşenleri girişin farklı sınıflara ait olma olasılığını verir. Çıktı çıkarımı daha sonra doğruluğu ölçmek için kayıp fonksiyonunda kullanılır. Bundan sonra, nöronlardaki her ağırlık için gradyan hesaplanır. Ağdaki katmanlar birbirinin üstünde olduğundan, çıktı katmanından girdi katmanına olan gradyanı hesaplamak için zincir kuralına sahip kısmi bir türev benimsenir. Daha sonra, ağırlıklar gradyan değeri kullanılarak güncellenir.

2.1. Evrişimsel Sinir Ağları

Evrişimsel Sinir Ağları algoritması, ses, görüntü ve video gibi veriler üzerinde evrişim işlemleri uygulayan bir tür sinir ağları algoritmasıdır. ConNN, basit sinir ağları gibi girdilerdeki tüm özellikleri kullanmak yerine, bir girdi girdilerinden yerel özellikleri çıkarmak için kullanılır. Bu nedenle, çıktı özellikleri ağırlıkları birlikte paylaşır, böylece basit sinir ağlarının kullanımına kıyasla bağlantı sayısı oldukça azalır. Basit bir sinir ağına benzer şekilde ConNN de, temel olarak bir girdi katmanı, gizli katman ve çıktı katmanından oluşur. Her katman, önceki katmandan girdi (girdi özellik haritası olarak da bilinir) alır ve ardından bir sonraki katmana çıktı üretir. Şekil 2.2 tipik bir ConNN yapısını göstermektedir.



Şekil 2.2. ConNN mimarisi örneği

Bu örnekte, girdi katmanı bir RGB görüntüsünün pikselleridir. ConNN'nin gizli katmanı, evrişimsel katman, havuzlama katmanı ve tam bağlantılı katman gibi çeşitli türlerde olabilir. Son olarak, çıktı katmanı, olasılık puanları veya regresyon değerleri olabilen nihai sonuçların üretilmesinden sorumludur.

2.1.1. Evrişimsel Katman

Evrişim, girdi boyunca küçük bir pencereyi (kernel ya da çekirdek olarak da bilinir) kaydırarak çıktı üretmek için çarpma-biriktirme (MAC) işlemini içerir. Algoritma, bir görüntüden anlamlı yerel özellikler çıkarmayı amaçlar. Kernel, ağırlıklar adı verilen bir dizi öğrenilebilir parametreden oluşan 2 boyutlu bir matris olarak düşünülebilir. Ağırlık parametreleri, gerçek çıktı ile istenen çıktı arasındaki hataya göre eğitim aşamasında ayarlanır. Evrişimsel katmanının çıktısı, girdi özellik haritaları ve filtrenin evrişim sonucudur. Çıktı şekli de 2 boyutlu bir matristir. Evrişimsel katmanı kavramını anlamak için, Eşitlik (2.1)'de tek boyutlu verilerle ayrık-zamanda evrişim ifadesi verilmiştir.

$$y[n] = x[n] * h[n] = \sum_{k=-\alpha}^{\alpha} x[n] \cdot h[n-k] \tag{2.1}$$

Burada x[n] girdiyi, h[n] dürtü yanıtını, y[n] çıktıyı, * işareti evrişim işlemini belirtmektedir.

Aynı kavram ile hem yatay hem de dikey yönlerde evrişim yoluyla 2D verilere evrişim uygulanabilir. Evrişim hesaplamasını göstermek için, y, w ve x sırasıyla çıktıları,

ağırlıkları (kerneldeki parametreler) ve girdileri göstersin. Dürtü yanıt işlevi Kernel ile değiştirilir. Y matrisinin m sütunu ve n satırı için konvolüsyon hesaplaması Denklem (2.2)'deki gibi ifade edilebilir.

$$y[n] = x[m,n] * w[m,n] = \sum_{j=-\alpha}^{\alpha} \sum_{i=-\alpha}^{\alpha} x[i,j] \cdot w[m-i, n-j] \quad (2.2)$$

Burada α, x'in elemanının indeksidir.

Bu matematiksel formülün hesaplamasını göstermek için, kernel, 2×2'lik bir matris olsun. Çıktı y[1,1] Denklem (2.3)'teki gibi ifade edilebilir.

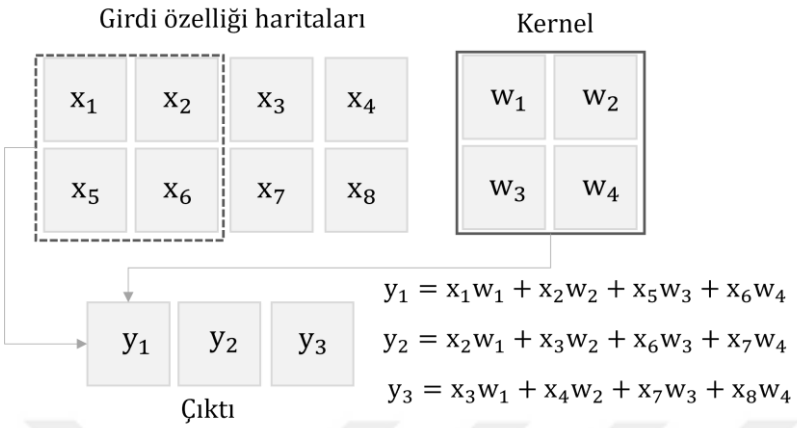
$$y[n] = x[0,0] \cdot w[1,1] + x[1,0] \cdot w[0,1] + x[0,1] \cdot w[1,0] + x[1,1] \cdot w[0,0] \quad (2.3)$$

Çıktının, girdi ile 180 derecelik bir açıyla çevrilmiş kernel ile çarpılması ve ardından tüm çarpanların toplanmasıyla hesaplandığı görülebilir. Bununla birlikte, gerçek dünya uygulamasında, çoğu sinir ağı çerçevesi, çevirme adımını azaltmak için çapraz korelasyon işlemiyle evrişim gerçekleştirir ve çapraz korelasyon, programlama ve bellek yönetimi açısından evrişim hesaplamasından daha kolaydır. Korelasyon Eşitlik (2.4)'teki gibi gösterilmiştir.

$$y[m,n] = \sum_{j=-\alpha}^{\alpha} \sum_{i=-\alpha}^{\alpha} x[i,j] \cdot w[m+i, n+j] \quad (2.4)$$

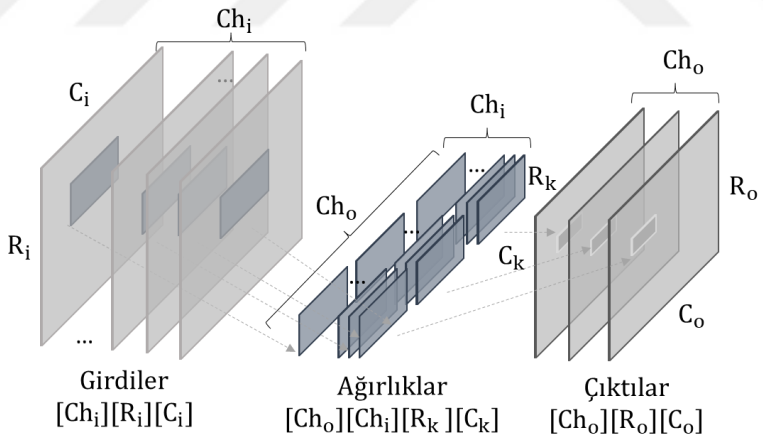
Çapraz korelasyon işlemi, evrişime çok benzer, ancak küçük bir fark vardır. Girdi ile çarpmadan önce kernel matrisini çevirmez. Ancak korelasyon çıktısı, evrişim çıktısının çevrilmiş bir versiyonudur. Bu nedenle, çıkarılan özellikler ters çevrildiğinde bir görüntünün özelliğini çıkarmak için çapraz korelasyon kullanmak kullanışlı değildir. Ancak, kernel matrisi öğrenilebilmiş ağırlık parametrelerinden oluştuğu için sinir ağı modelinde evrişim yerine çapraz korelasyon işlemi kullanılabilir. Ağırlıklar eğitim sırasında güncellenir, böylece elemanın kerneldeki konumu çıktı sonuçlarını etkilemez. Bu nedenle çapraz korelasyon işlemi ile evrişim katmanı uygulanabilir. Şekil 2.3 girdi

matris boyutunun 4×2 ve kernel boyutunun 2×2 olduğu evrişimsel katmandaki işlemlerin bir örneğini göstermektedir. Bir kaydırma adımı boyutu ile sonuç 3×1 çıktı matrisidir.



Şekil 2.3. 2x4 matris girdi ve 2x2 matris kernel üzerinde evrişim örneği

Evrişimsel katmanında kernel sayısı birden fazla olabilmektedir. Girdi/çıktı özellik haritalarının yapısı ve ağırlıklar çok boyutlu bir dizi olarak kabul edilmiştir. Şekil 2.4 konvolüsyonel katmanındaki veri yapısı boyutlarını göstermektedir.



Şekil 2.4. Evrişim katmanında Girdilerin, çıktıların ve ağırlıkların yapısı

Burada C_i, R_i ve Ch_i'nin girdi özelliği eşlemelerinin sütunlarını, satırlarını ve kanallarını gösterdiği yerdedir. K, kernel'in boyutunu ve C_o, R_o ve Ch_o, çıktı özelliği haritalarının sütunlarını, satırlarını ve kanallarını belirtmektedir. Evrişimsel katmanı Şekil 2.5'te sözde kod olarak uygulanabilmektedir.

Algoritma 1: Evrişim döngüleri

Girdi: Girdiler, Ağırlıklar, Yanlılık
Çıktı: Çıktılar
for $ch_o \leftarrow 0$ **to** Ch_o **do**
  for $r_o \leftarrow 0$ **to** R_o **do**
    for $c_o \leftarrow 0$ **to** C_i **do**
      for $ch_i \leftarrow 0$ **to** Ch_i **do**
        for $r_k \leftarrow 0$ **to** R_k **do**
          for $c_k \leftarrow 0$ **to** C_k **do**
            Çıktılar $[ch_o][r_o][c_o]$ +=
            Girdiler $[ch_i][S\times r_o + r_k][S\times c_o + c_k] \times$
            Ağırlıklar $[ch_o][ch_i][r_k][c_k]$
      Çıktılar $[ch_o][r_o][c_o]$ + Yanlılık $[ch_o]$

Şekil 2.5. Evrişimsel hesaplamasının sözde kodu

Girdi ve ağırlıklar üzerinde evrişim için 6 iç içe döngü gereklidir. En fazla 2 iç döngü, kernel'de eleman indeksleme için kullanılır. En dıştaki döngü kernel'in kanalı için kullanılır. Dolgu tekniği, girdi özelliği haritalarının sınırlarını genişletmek için kullanılır. Adım, kernel için kaydırma adımı boyutunu tanımlamak için kullanılır. Verilen P, S ve K değerleri için çıktının satır ve sütun sayısı şu formüllerden belirlenebilir: $C_o = \frac{C_i - K + 2P}{S} + 1$ ve $R_o = \frac{R_i - K + 2P}{S} + 1$.

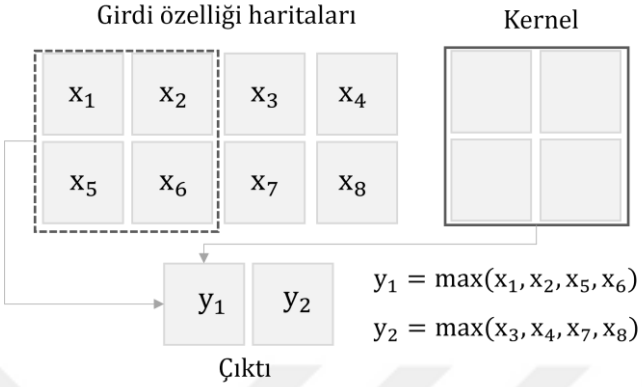
2.1.2. Aktivasyon Katmanı

Tipik olarak, aktivasyon veya doğrusal olmayan katman, evrişim FC katmanındaki her bir katmandan sonra da uygulanabilir. Aktivasyon yönteminin sezgisi, çıktılar üzerinde düşük etkileri olan girdi özellik haritalarını filtrelemektir. Aktivasyon fonksiyonu için Sigmoid, Doğrultulmuş Lineer Birim (ReLU) ve Tanh gibi birkaç doğrusal olmayan yöntemler tanıtılmıştır. Son zamanlarda ReLU aktivasyonu, hızlı yakınsama sonuçları sağladığı için diğer tekniklere göre daha sık kullanılır.

2.1.3. Havuzlama Katmanı

Bu yöntem, önemli özellikleri bozmadan girdi özellik haritalarının uzaysal boyutunu küçültmek için yaygın olarak kullanılır. Maksimum havuzlama (Maxpool), Ortalama havuzlama (Ort. havuz) veya L2-norm havuzlama gibi çıktılar oluşturmak için çeşitli

teknikler vardır. Havuzlama katmanının öğrenilebilir parametresi yoktur. Bu nedenle, işlem sayısını azaltmak için havuzlama yöntemi kullanılır. Şekil 2.6 Maksimum havuzlama katmanının bir örneğini göstermektedir.



Şekil 2.6. Kaydırma adımı 2 olduğunda maksimum havuzlama örneği

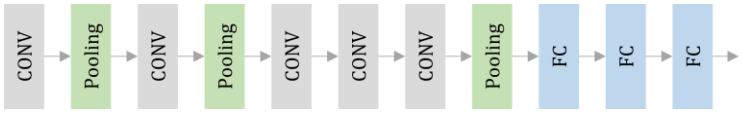
Bu örnekte, girdi özellik haritaları 4×2 matristir ve kernel boyutu 2×2'dir. Kaydırma adımı 2 olduğunda, sonuçlar 2×1 çıktı matrisidir. Havuzlama boyutu (K) ve kaydırma adımı (S), Havuzlama katmanı için hiper parametrelerdir. Verilen S ve K değerleri için çıktının satır ve sütun sayısı şu şekilde hesaplanabilir: $C_o = \frac{C_i - K}{S} + 1$ve $R_o = \frac{R_i - K}{S} + 1$.

2.1.4. Tam Bağlantılı Katman (FC)

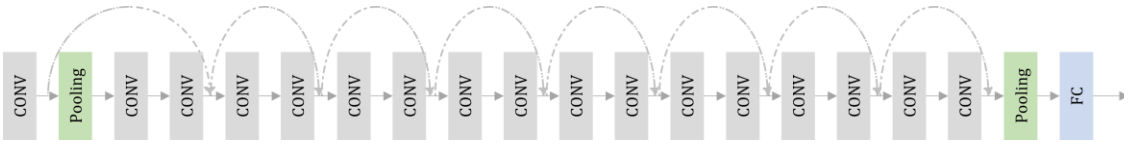
FC katmanı önceki katmandaki tüm örnekleri çıktılara bağlayan basit bir sinir ağı katmanıdır. Bu nedenle, bu katman, tüm özellikleriyle sonucu temsil etmek için tipik olarak modelin son katmanı olarak sunulur. FC katmanı tipik olarak daha az uzaysal boyuta sahiptir. Sonuç olarak, hesaplama gereksinimleri evrişimsel katmanına kıyasla daha düşüktür.

2.2. ConNN Mimarisi

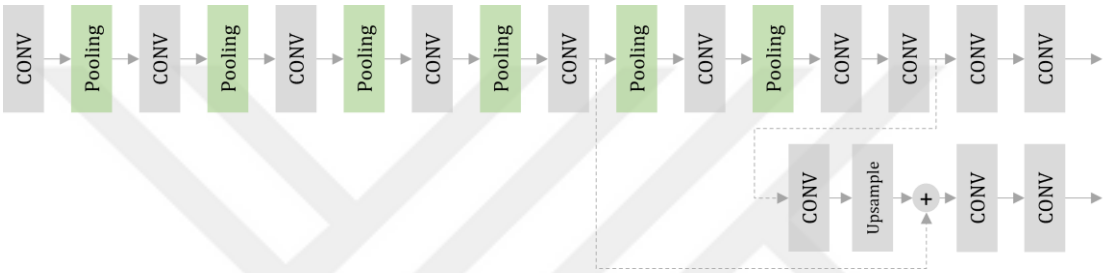
Bu kısımda, ikisi görüntü sınıflandırma için kullanılan ve biri nesne algılama için tasarlanmış üç adet ConNN modeli ve ConNN mimarisi gösterilmektedir. Bu modellerin farklı sayıda katmanları vardır ve hesaplama kabiliyeti gereksinimleri farklıdır. Ayrıca her modelde kullanılan bağlantı tekniği de farklıdır.



(a)



(b)



(c)

Şekil 2.7. Görüntü sınıflandırma ve nesne algılama için yaygın olarak kullanılan ConNN algoritmalarının ağ yapıları. (a) AlexNet modelinin yapısı (b) ResNet-18 modelinin yapısı (c) YOLOv3-tiny modelinin yapısı

AlexNet (Krizhevsky ve diğ., 2012), 2012 yılında ILSVRC galibi olarak tanıtıldı. Bu model, klasik makine öğrenimi algoritmalarına kıyasla sınıflandırma doğruluğunu önemli ölçüde artıran ilk ConNN modelidir. AlexNet ‘in ImageNet veri setindeki en iyi 5 doğruluğu %84,6'dır. AlexNet, Şekil 2.7.(a)'da gösterildiği gibi 5 evrişim katmanından, 3 havuzlama katmanından ve 3 FC katmanından oluşur. AlexNet 61 milyon öğrenilebilmiş ağırlığa sahiptir ve 2,2 milyar çarpma ve biriktirme işlemi gerektirmektedir. Girdi özellik haritalarının boyutları ve ağırlıkları Tablo 2.1'de listelenmiştir. ResNet (He ve diğ., 2016), ILSVRC 2015 sınıflandırma zorluğunda görüntü algılamada birinci sırayı alarak %89,2'den yüksek ve %95,51'e varan ilk 5 doğruluğu sağlamıştır. Yazarlar, farklı sayıda katman kullanan ancak aynı kavramda sahip çeşitli modeller sunmuştur. Bu model, geri yayılım sürecinde gradyan kaybolma problemiyle başa çıkmak için iki bitişik olmayan katman arasındaki atlama bağlantılarını kullanmıştır. ResNet-18'in mimarisi Şekil 2.7.(b)'de sunulmaktadır. ResNet-18, 17 evrişim katmanı, 1 FC katmanı ve 2 havuzlama

katmanından oluşmuştur. Yaklaşık 11 milyon ağırlığa sahiptir. ResNet-18'deki veri boyutları ve ağırlıkları Tablo 2.2'de sunulmaktadır.

Tablo 2.1. AlexNet ‘in katman yapıları ve parametreleri

Katman	Tip-adımlama-dolgu	Kernel boyutu $(W_k \times H_k)/\#kernel$	Girdi boyutu $(W_{in} \times H_{in})/C_{in}$	İşlemler FLOPs $\times 10^9$
1	Conv-4-0	(11 × 11) / 96	227 × 227 × 3	0.211
2	Maxpool	3 × 3	55 × 55 × 96	-
3	Conv-1-1	(5 × 5) / 256	27 × 27 × 96	0.896
4	Maxpool	3 × 3	27 × 27 × 256	-
5	Conv-1-1	(3 × 3) / 384	13 × 13 × 256	0.299
6	Conv-1-1	(3 × 3) / 384	13 × 13 × 384	0.449
7	Conv-1-1	(3 × 3) / 256	13 × 13 × 384	0.299
8	Maxpool	3 × 3	13 × 13 × 256	-
9	FC		9216	0.075
10	FC		4096	0.03
11	FC		4096	0.008

Tablo 2.2. ResNet-18'in katman yapıları ve parametreleri

Katman	Tip-adımlama-dolgu	Kernel boyutu $(W_k \times H_k)/\#kernel$	Girdi boyutu $(W_{in} \times H_{in})/C_{in}$	İşlemler FLOPs $\times 10^9$
1	Conv-2-1	(11 × 11) / 96	227×227×3	0.211
2	Maxpool-2	(2 × 2)	128×128×64	-
3	Conv-1-1	(3 × 3) / 64	64 × 64 × 64	0.302
4	Conv-1-1	(3 × 3) / 64	64 × 64 × 64	0.302
5	Conv-1-1	(3 × 3) / 64	64 × 64 × 64	0.302
6	Conv-1-1	(3 × 3) / 64	64 × 64 × 64	0.302
7	Conv-2-1	(3 × 3) / 128	64 × 64 × 64	0.151
8	Conv-1-1	(3 × 3) / 128	32 × 32 × 128	0.302
9	Conv-1-1	(3 × 3) / 128	32 × 32 × 128	0.302
10	Conv-1-1	(3 × 3) / 128	32 × 32 × 128	0.302
11	Conv-2-1	(3 × 3) / 256	32 × 32 × 128	0.151
12	Conv-1-1	(3 × 3) / 256	16 × 16 × 256	0.302
13	Conv-1-1	(3 × 3) / 256	16 × 16 × 256	0.302
14	Conv-1-1	(3 × 3) / 256	16 × 16 × 256	0.302
15	Conv-2-1	(3 × 3) / 512	16 × 16 × 256	0.151
16	Conv-1-1	(3 × 3) / 512	8 × 8 × 512	0.302
17	Conv-1-1	(3 × 3) / 512	8 × 8 × 512	0.302
18	Conv-1-1	(3 × 3) / 512	8 × 8 × 512	0.302
19	FC		1000	0.001

You Only Look Once (YOLO) (Redmon ve diğ., 2016), çeşitli nesneleri gerçek zamanlı olarak algılayabilen hızlı bir nesne algılama algoritması olarak tanıtılmıştır. Diğer yöntemler gibi tipik bir kaydırma adımı pencere ile nesneleri algılamak yerine, bu yöntem regresyon problemi olarak bir algılama görevi gerçekleştirmiştir. YOLO, sınırlayıcı kutuyu tahmin etmeye çalışır ve aynı zamanda sınıflandırma sonuçları üretir. Yazarlar YOLO'yu YOLOv3 (Redmon ve diğ., 2018) olarak güncellemiştir.

Şekil 2.7.(c), YOLOv3'ün mimarisini küçük bir versiyonda açıklamaktadır. Bu model 13 evrişim katmanından ve 6 havuzlama katmanından oluşmuştur. YOLOv3-tiny, yaklaşık 8,8 milyon kernel ağırlığına sahiptir ve 5,56 milyar çarpma ve biriktirme işlemi gerektirmiştir. YOLOv3-tiny yapısının detayları Tablo 2.3'te listelenmiştir.

Tablo 2.3. YOLOv3-tiny'nin katman yapıları ve parametreleri

Katman	Tip-adımlama-dolgu	Kernel boyutu $(W_k \times H_k)/\#kernel$	Girdi boyutu $(W_{in} \times H_{in})/C_{in}$	İşlemler FLOPs $\times 10^9$
1	Conv-1-1	(3 × 3) / 16	416×416×3	0.150
2	Maxpool-2	(2 × 2)	416×416×16	-
3	Conv-1-1	(3 × 3) / 32	208×208×16	0.399
4	Maxpool-2	(2 × 2)	208×208× 32	-
5	Conv-1-1	(3 × 3) / 64	104×104×32	0.399
6	Maxpool-2	(2 × 2)	104×104×64	-
7	Conv-1-1	(3 × 3) / 128	52×52×64	0.399
8	Maxpool-2	(2 × 2)	52×52×128	-
9	Conv-1-1	(3 × 3) / 256	26×26×256	0.399
10	Maxpool-2	(2 × 2)	26×26×256	-
11	Conv-1-1	(3 × 3) / 512	13×13×256	0.399
12	Maxpool-2	(2 × 2)	13×13×512	-
13	Conv-1-1	(3 × 3) / 1024	13×13×512	1.595
14	Conv-1-1	(3 × 3) / 256	13×13×1024	0.089
15	Conv-1-1	(3 × 3) / 512	13×13×256	0.399
16	Conv-1-1	(3 × 3) / 255	13×13×512	0.044
17	Conv-1-1	(3 × 3) / 128	13×13×256	0.011
18	Conv-1-1	(3 × 3) / 256	13×13×384	1.196
19	Conv-1-1	(3 × 3) / 255	13×13×256	0.088

2.3. FPGA Üzerinde ConNN'nin Hızlandırılması

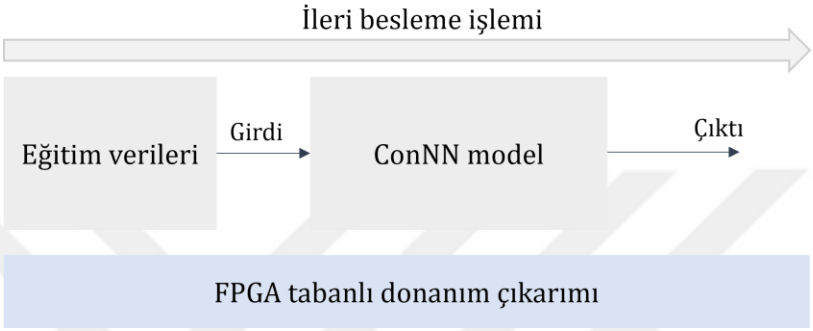
2.3.1. FPGA'ya Girişi

Alan Programlanabilir Kapı Dizileri (FPGA) bir süredir birçok alanda tanıtılan ancak henüz popüler olmayan bir cihazdır. Ancak son zamanlarda, uyabilen bilgi işlem yetenekleri, yüksek performanslı bilgi işlem için artan talebe yanıt olarak bir donanım eğilimi haline gelmiştir. Böylece FPGA, donanım hızlandırmada yüksek ilgi görmüştür. Bu eğilimi günümüz FPGA pazarında gözlemleyebilmekteyiz. İki ana FPGA üreticisi vardır: Xilinx ve Altera. 2015 yılında Altera, en büyük çip üreticisi olan Intel'in bir parçası olmuştur. Daha sonra Xilinx, çip endüstrisindeki en büyük satın alma ile 2020 yılında AMD tarafından satın alınmıştır. Bu hamlelerle birlikte, FPGA'nın bir donanım bilgi işlem platformu olarak daha çılgınca kullanılacağı görülmüştür. FPGA, belirli uygulamalar için esnek bir donanım platformu sağlamak üzere yeniden yapılandırma yeteneğine sahip özel bir mantık cihazıdır. FPGA'nın ana ayırt edici özellikleri paralellik, düşük güç tüketimi ve tasarım esnekliğidir. FPGA, programlanabilir bir ara bağlantı ağı kullanılarak bağlanan bir dizi yapılandırılabilir mantık bloğundan (CLB'ler) oluşmuştur. Bu nedenle, CLB'leri yapılandırarak ve ara bağlantıları yönlendirerek FPGA'nın üretimden sonra yeniden programlanmasına izin vermektedir. Ayrıca, FPGA, sayısal işaret işleme (DSP), bir dizi arama tablosu (LUT), bir Flip-flop (FF) ve blok RAM (BRAM) olarak adlandırılan bir çip üstü bellek gibi bir dizi entegre bileşene sahiptir.

FPGA üzerinde özel bir devre oluşturmak için klasik bir FPGA tasarımı, VHDL ve Verilog gibi Donanım Tanımlama Dilleri (HDL) şemalarını kullanır. Son birkaç yılda, Xilinx, C ve C++ dilleri gibi yüksek düzeyli programlama dilleri ile kullanımı kolaylaştırmak için sentez araçları sağlamış, böylece FPGA üzerinde uygulama kolay ve hızlı hale gelmiştir. Tasarım süreci, bağlantılarla birlikte istenen modülleri içeren entegre bir tasarımın oluşturulmasıyla başlar. Bundan sonra, sentez süreci, ara bağlantı ve mantık yapılandırmasından oluşan bir ara bağlantı dosyası oluşturmak için kullanılır. Bir sonraki adım, konfigürasyonları cihazdaki mevcut kaynaklarla eşleştirir. Daha sonra, cihaz üzerinde son donanım konfigürasyonunu oluşturmak için yerleştir ve yönlendir işlemleri gerçekleştirilir. Son olarak, bit akışı oluşturma işlemi, karta programlamak için bit akışı dosyalarını üretmek için kullanılır.

2.3.2. FPGA Tabanlı Hızlandırıcı

Donanım, ConNN hesaplamasında önemli bir rolü oynamaktadır. Son zamanlarda, güç verimliliği ile hesaplamayı hızlandırmak için FPGA üzerinde ileri besleme işlemi ve eğitimi üzerinde çalışılmaktadır. Ancak küçük ölçekli FPGA için kaynak sınırlaması nedeniyle sadece çıkarım yapabilmektedir. Şekil 2.8 ConNN hızlandırması için uç FPGA'nın kullanımını göstermektedir.



Şekil 2.8. FPGA'da ConNN çıkarımının hızlandırılması

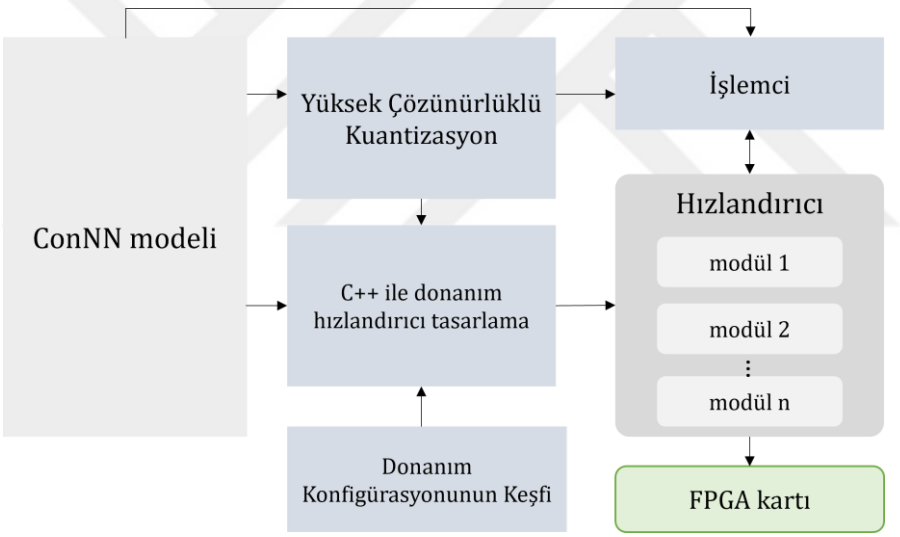
FPGA test verilerini yükler, ConNN girişine gönderir ve ardından çıkarım çıkışını gerçekleştirir. Veri depolama ve bilgi işlem işlemcisi ana donanım gereksinimleridir. Veri depolama hem girdi verilerini hem de ağırlıkları tutmak için kullanılır. Bilgi işlem işlemcisi, çarpma ve biriktirme yapmak için kullanılır. FPGA, sınırlı bir çip üzerinde bellek ve az miktarda bilgi işlem işlemcisi sağlamaktadır. Bu nedenle, kaynak sınırlaması, FPGA'da ConNN modellerinin uygulanmasında bir zorluk teşkil etmektedir. Zorluğu gidermek için çeşitli teknikler sunulmuştur. Yoğun bir veriyle başa çıkmak için verimli bir veri akışı ve işlem hattı gereklidir. Paralel bilgi işleme olmalı genellikle bilgi işleme gecikmesini azaltmak için kullanılır. Ek olarak, veri sıkıştırma yaygın olarak 32 biti daha düşük bitlere dönüştürmek için kullanılır, böylece bilgi işlemsel karmaşıklığı ve bellek gereksinimi azalır.

2.4. Motivasyon

Birkaç FPGA tabanlı ConNN hızlandırması, ConNN'nin büyük ölçekli FPGA'daki performansını iyileştirmeye odaklanmıştır. (Z. Liu ve diğ., 2017), evrişimin çalışma zamanına dayalı optimal bir donanım konfigürasyonu sunmuştur. Çok yüksek hesaplama

gereksinimlerine ulaşmak için ConNN modelinin Cloud FPGA'larına eşlenmesi (Chen ve diğ., 2019; Fowers ve diğ., 2018) için de gösterilmiştir.

Ancak, küçük ölçekli FPGA'da ConNN hızlandırması, sınırlı sayıda kaynak nedeniyle kaynak verimliliğine daha fazla dikkat gerektirmektedir. Küçük ölçekli FPGA çok düşük güç tüketimi sağlamasına rağmen, kaynak kısıtlamaları nedeniyle işlem oranı sınırlıdır. ConNN modelinin düşük maliyetli FPGA'da eşlenmesi hem yazılım hem de donanımın verimli bir şekilde tasarlanmasını gerektirmektedir. Kaynak verimliliğine dayalı optimal donanımı keşfetmenin dikkate alınması gerekir. Eş zamanlı olarak, kaynak gereksinimlerinin sayısını azaltmak için modelin yazılım tabanlı sıkıştırılması gerekir. Bu nedenle, bu çalışmada, ConNN için hızlandırıcıyı küçük ölçekli FPGA'da uygulamak için bir donanım ve yazılım ortak tasarımı araştırılmıştır. Bu tezin motivasyonu Şekil 2.9 'da sunulmaktadır.



Şekil 2.9. Tez motivasyonu: yazılım ve donanım ortak tasarımı kullanılarak küçük FPGA'da ConNN'nin hızlandırılması

Kaynak kısıtlamaları altında en uygun konfigürasyonu elde etmek için bir donanım yapısı araştırması yapılmıştır. Hızlandırıcıyı C++ programlama ile uygulamak için C++ tabanlı bir sentez derleyicisi kullanılmıştır. Kaynak gereksinimlerini optimize etmek için yüksek çözünürlüklü logaritmik nicemleme yöntemi sunulmaktadır. Hızlandırıcımız, çalışma zamanı sırasında katmanın karakteristiğine dayalı olarak donanımın farklı mimarilerle yeniden yapılandırıldığı uyarlanabilir yeteneklere sahiptir. Hızlandırıcı, ConNN çıkarımını gerçekleştirmek için işlemciyle birlikte çalışmaktadır.

2.5. Literatür İncelemesi

Bu kısımda, FPGA'larda ConNN'nin hızlandırılması için önerilen en güncel yöntemler tanıtılmıştır. İnceleme, tasarım zorluklarına dayalı olarak üç bölümde sunulmaktadır: donanım mimarisi tasarımı, donanım için bilgi işlem optimizasyonu ve ConNN model sıkıştırma teknikleri.

2.5.1. FPGA Tabanlı Hızlandırıcı Mimarisi

ConNN'nin hızlandırılması için farklı donanım tasarım teknikleri önerilmiştir. (C. Zhang ve diğ., 2015) hesaplama modüllerinden ve bir bellek alt sisteminden oluşan bir ConNN hızlandırıcısı önermiştir. Yazarlar, hızlandırıcı için donanım yapısı tasarlamak adına işlem oranı performansı ve çip dışı bellek bant genişliğini ilişkilendiren çatı modelini benimsemiştir. (Guo ve diğ., 2017) 3×3 kernel boyutu ile evrişim hesaplama için paralel işleme elemanları ve çip üzerinde arabellek sunmuştur. Diğer kernel boyutları da desteklenmiş ancak kernel'e doldurma ve ekleme gerektirmiştir. Hızlandırıcı, çip üzeri kullanımıyla, bağımsız evrişim hesaplama için giriş özelliği eşlemelerini tüm işleme elemanı (PE) ile paylaşabilmiştir.

(J. Zhang ve diğ., 2017) OpenCL standardını kullanarak ConNN'nin hızlanmasını göstermiştir. Hızlandırıcı, 2D evrişim adına verileri yeniden düzenlemek için bir hat arabelleği kullanan yerel bir bellek alt sistemi olan 2D PE'lerden ve bilgi işleme görevlerini bölmek ve zamanlamak için bir 2D göndericiden oluşmuştur. Hat arabelleği yapısı da (L. Lu ve diğ., 2017) içinde sunulmuştur. Hızlandırıcıları bir hat arabellek yapısı ve bir Winograd PE ünitesinden oluşmuştur. Veri aktarımı ve bilgi işlem çakıştırmak için, girdiler PE tarafından okunurken, sonraki girdi hatları ilk giren ilk çıkar (FIFO) yöntemi kullanarak girdileri bellekten yüklemiştir. PE ünitesi, paralellik elde etmek için döngü açma yöntemi kullanılarak tasarlanmıştır.

Matris çarpması ve veri düzenleyici modüllerinden oluşan bir donanım hızlandırıcı (Guan ve diğ., 2017) içinde sunulmuştur. Matris çarpma modülü, küçük girdi ile çarpma işlemi gerçekleştirmek için tasarlanmıştır. Veri çakışması yoluyla verimi artırmak için pingpong tarzı kullanan girdiler için çift arabelleği uygulanmıştır. Veri düzenleyici, verileri harici bellekten matris çarpma modülüne eşlemek için uygulanmıştır. (Wei ve diğ., 2017)

ConNN için otomatikleştirilmiş bir akış uygulamasıyla bir sistolik dizi yapısı önermiştir. 2D sistolik dizi, PE bloklarından oluşmuştur. PE, her döngüde ağırlığı ve girdi verilerini yatay ve dikey olarak bitişik PE'lere kaydırmıştır. Yazarlar, tasarımın yönlendirme karmaşıklığını azalttığını ve FPGA düzeninin 2D yapısına uyacak şekilde tasarlandığından zamanlama kısıtlamalarına ulaştığını iddia etmektedir.

Sistolik dizi tabanlı hızlandırıcı da (W. Zhang ve diğ., 2020) içinde önerilmiştir. Her sistolik dizi, veri yeniden kullanım oranını iyileştirmek için kendi arabelleğine sahip PE'lerden oluşur. Sistolik dizinin PE'si, bir seferde belirli bir boyuta sahip vektörün toplamını gerçekleştirmek adına bir akümülatöre sahiptir. Bellek bant genişliği gereksinimlerini karşılamak için verileri önceden getirmek ve verilerin yeniden kullanım oranlarını artırmak için üç seviyeli önbellekler sunulmuştur. Evrişimsel katman, havuzlama katmanı, normalleştirme ve tam bağlantılı katman için donanım modüllerini içeren entegre bir sistem kullanarak ConNN'nin hızlandırılmasını sunulmuştur. CONV modülü, kontrol mantığı, paralel toplayıcı ağaçları ve ReLU aktifleştirme bloklarından oluşmuştur. Paralel hesaplamaları etkinleştirmek için çarpma ünitesi, farklı katmanlar arasında donanım kullanımını iyileştirmek için her katmanın konfigürasyonuna dayalı açılma döngüsü ile oluşturulmuştur.

2.5.2. Evrişim Hızlandırma Yöntemi

ConNN algoritmasının hesaplamalarını hızlandırmak için girdi özellik haritaları ve ağırlıklar üzerinde bilgi işlemsel dönüşüm yöntemleri kullanılmıştır. Genel matris çarpımları (GEMM) ve Winograd algoritması olmak üzere FPGA'da evrişim hesaplamalarını optimize etmek için oldukça sık kullanılan iki yöntem vardır.

GEMM tekniği, ConNN çıkarımının işlem oranı performansını iyileştirmek ve bellek erişim fazlalığını (Kim ve diğ., 2017) azaltma doğrultusunda genellikle CPU ve GPU için derin öğrenme çerçevelerinde uygulanmıştır. Ek olarak, GEMM sadece evrişimsel katmanı değil, aynı zamanda FC katmanını da gerçekleştirmek için kullanılabilmektedir. Birkaç çalışma, GEMM'i destekleyen ConNN hızlandırıcısını uygulamıştır (S. E. Chang ve diğ., 2021; Ma ve diğ., 2018; Wei ve diğ., 2017). Evrişimi hızlandırmak için sistolik dizi tabanlı bir ConNN hızlandırıcısı (W. Zhang ve diğ., 2020) içinde sunulmuştur.

GEMM yöntemi, matris olarak 2 boyutlu dizi gerektirmektedir. Bu nedenle, bellek yönetimi ve verimli veri akışı önemli faktörlerdir. Girdi matrisini ana kanal şeklinde düzleştirip yeniden düzenleyerek matris çarpımını kolaylaştırmak için (Guan ve diğ., 2017) içinde bir veri düzenleyici tanıtılmıştır. Başka bir çalışma (Luo ve diğ., 2020), matris çarpımları için toplu seviye paralelliği geliştirmek için kanal-yükseklik-genişlik-toplu (CHWB) modeli sunmuştur. GEMM çekirdeği, evrişimsel ve FC katmanlarının çalışmasını desteklemek için geliştirilmiştir. Hem ileri besleme hem de geri yayılım işlemleri için kullanılabilmektedir.

ConNN hesaplamalarını optimize etmenin başka bir yöntemi de Winograd minimum filtrelemedir. Bu algoritma, evrişim hesaplamasında çarpma sayısını azaltmayı amaçlamaktadır. Sonuç olarak, bu yöntem, GEMM'e (Lavin ve diğ., 2016) kıyasla işlem oranı performansında bir gelişme sağlamaktadır. İlk olarak, Winograd şeması girdi özellik eşlemeleri ve ağırlıkları Winograd alanına dönüştürür ve daha sonra dönüştürülmüş girdiler ve ağırlıklarla eleman bazında matris çarpımı gerçekleştirir. Yöntem daha sonra çarpma sonucunu ters olarak evrişim çıktısı olacak şekilde dönüştürür. Birçok çalışma, FPGA'da Winograd algoritmasını kullanarak evrişim hızlandırılmasını araştırmıştır. (L. Lu ve diğ., 2017), evrişim hesaplaması için hem yüksek paralellik hem de veri yeniden kullanım oranı elde etmek için bir 2D Winograd algoritması kullanmaktadır. ConNN hesaplamak için Winograd algoritmasını kullanan bir hızlandırıcı sunulmuştur (Aydonat ve diğ., 2017). Bu hızlandırıcı, PE'lerden ve arabelleklerinden oluşmuştur. PE'de nokta çarpım üniteleri, toplayıcılar ve önbellekler vardır. Her nokta çarpım ünitesi, Winograd dönüştürülmüş girdi özellikleri ve ağırlıkları ile çarpma ve biriktirme gerçekleşmiştir.

Ancak Winograd şeması, kaydırma adımı bir olduğunda ve kernel boyutu küçük olduğunda (3×3'e eşit veya daha küçük) kullanışlı bir tekniktir. Başka bir deyişle, 3×3'ten büyük kernel boyutlarına veya hatta 1×1 kernel boyutuna (hiçbir veri çakışması olmadığından) sahip ConNN modelleri, Winograd algoritmasını kullanmak için uygun değildir.

2.5.3. ConNN Sıkıştırma Yöntemi

Önceki kısımda, ConNN hızlandırması için FPGA'yı kullanan donanım tabanlı teknikler incelenmiştir. Bu kısım, FPGA'da ConNN hızlandırması için yazılım tabanlı tekniklere odaklanmıştır. Veri sıkıştırma, uygulama karmaşıklığını azaltmak için etkili tekniklerden biridir. ConNN modelleri, 32-bit kayan nokta veri gösterimi temelinde tasarlanmıştır. Ancak, FPGA'da kayan nokta kesinliği, çarpma ve veri depolama için yoğun kaynaktır. Sayısal kesinliği azaltmak için nicemleme yöntemi hem girdi özellik eşlemelerine hem de ağırlıklara uygulanmıştır. Ek olarak FPGA, kullanıcıların tamsayı ve kesirli kısımlar için belirli bit genişliğini belirleyebildiği, sabit nokta hassasiyeti olarak bilinen özel bir bit genişliği gösterimini desteklemiştir.

ConNN çalışmalarının FPGA tabanlı hızlandırması, donanım tüketimini azaltmak ve verim performansını artırmak için sabit noktalı veri türünü kullanmıştır (Feng ve diğ., 2016; Lo ve diğ., 2020; Véstias ve diğ., 2018). ConNN modelindeki katman yapısı farklı olduğu için (S. E. Chang ve diğ., 2021; Qiu ve diğ., 2016), optimal bir bit genişliği sağlamak için çoklu hassasiyet tabanlı ConNN modelleri sunmuş, bu da yüksek kaynak kullanımı ve verimli Bellek bant genişliği kullanımı sağlamıştır. Bellek gereksinimini daha da azaltmak için ağırlıkları 1-bitlik veri formatında (Courbariaux ve diğ., 2016) temsil etmek için ConNN modeline ikilileştirme yöntemi uygulanmıştır. Bu çalışmada ağırlık gerçek bir değer olarak hafızaya kaydedilmekte ve ileri besleme ve geri yayılım hesaplamaları sırasında ağırlık sadece -1 veya 1 olarak ikili hale getirilmektedir.

İkilileştirmenin AlexNet'e uygulanması yalnızca 7,4 MB gerektirir. Böylece tüm model çip üzerindeki bellekte saklanabilmiştir. Sonuçlar, hızlandırıcının bellek bant genişliği sınırı (Umuroglu ve diğ., 2017) olmaksızın yüksek performansa ulaştığını göstermiştir. Çarpan kullanmanın maliyetini daha da azaltmak için (S. E. Chang ve diğ., 2021; Madadum ve diğ., 2022; Vogel ve diğ., 2018) içinde logaritmik nicemleme yöntemi sunulmuştur. Logaritmik nicemleme, veri formatını kayan nokta verisinden logaritmik dayalı veriye dönüştürür. Bu nedenle, çarpma işlemleri bit kaydırıcı ve toplama işlemleri ile değiştirilebilir. Bu yöntem, FPGA'da DSP kullanım sayısını önemli ölçüde azaltmıştır.

3. EVRİŞİMSEL SİNİR AĞLARININ HIZLANDIRILMASI

Bu bölümde, FPGA üzerinde ConNN'nin hızlandırılması için donanım tabanlı teknikleri bahsedilmiştir. 3-boyutlu verilerle çarpma ve toplamayı içeren evrişim döngüsünü basitleştirmek için döngü optimizasyon yöntemleri sunulmuştur. Evrişimin hızlanmasını analiz etmek için döngü optimizasyon tekniklerinin sayısal karakterizasyonu sağlanmıştır. Kaynak açısından verimliliğe sahip bir hızlandırıcı anlatılmıştır. Son olarak, hızlandırıcı performansı ve diğer çalışmalarla karşılaştırması verilmiştir.

3.1. Evrişim Döngü Optimizasyonu

Evrişim katmanındaki özellik haritaları, 3-boyutlu bir veri yapısında düzenlenir. Bu nedenle, evrişim sonuçlarını elde etmek için ağırlıklarla birlikte girdi özellik haritaları arasında çarpma-biriktirme işlemlerinden oluşan 6 seviyeli iç içe for döngüsü gerekmektedir. Evrişim hesaplamasını optimize etmek için genellikle genel matris çarpımı (General Matrix Multiplication: GEMM) kullanılmıştır (Ma ve diğ., 2018; Wei ve diğ., 2017). GEMM kullanılarak yapılan evrişim hesaplaması C = AB + C olarak tanımlanmıştır. Burada A ağırlık matrisi, B girdi matrisi ve C çıktı matrisidir. Bu hesaplamayı gerçekleştirmek için 3-boyutlu veriler 2-boyutlu bir matrise dönüştürülmektedir. Bu nedenle, girdi özellik haritaları, K×N boyutunda bir iki-boyutlu matrise dönüştürülür, yani bu girdi matrisinin K satırı ve N sütunu vardır. Benzer şekilde, çıktı özellik haritaları ve ağırlık sırasıyla M×N ve M×K matrisleri olacak şekilde ayarlanmıştır. Bu şekilde, parametrelerin boyutu azaldıkça evrişim döngülerinin seviyesi azaltılabilmektedir. Şekil 3.1 GEMM algoritmasına dayalı evrişim döngülerinin sözde kodunu göstermektedir. Bu hesaplamayı FPGA üzerinde uygulamak için, döngü döşeme ve döngü açma dahil olmak üzere döngü optimizasyon teknikleri tanıtılacaktır.

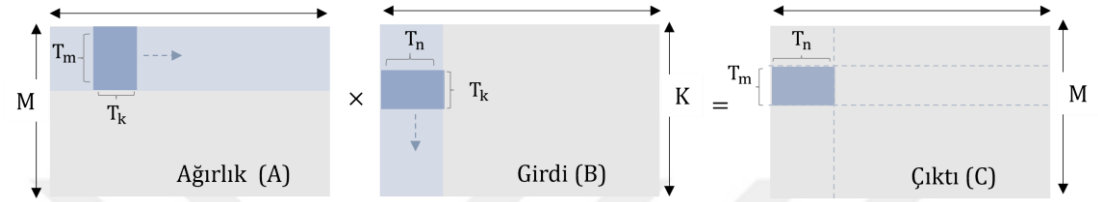
Algoritma 2: Evrişim hesaplaması

Girdi: A, B
Çıktı: C
for m ← 0 **to** M **do**
  for k ← 0 **to** K **do**
    for n ← 0 **to** N **do**
      $C_{m,n} = A_{m,k} \times B_{k,n} + C_{m,n}$

Şekil 3.1. GEMM-tabanlı evrişim döngülerinin sözde kodu

3.1.1. Döngü Döşeme

FPGA, tüm parametreleri depolayamayan sınırlı bir belleğe sahiptir. Önceki birçok çalışma, bellek kısıtlamaları sorununu ele almak için bir döngü döşeme yöntemi uygulamıştır (Ma ve diğ., 2018; Wei ve diğ., 2017). Döngü döşeme yöntemi, işlemcinin yalnızca küçük miktarlardaki verileri hesaplayabilmesi için yürütmeyi aşamalara ayırma tekniğidir. Döngü döşeme kavramı Şekil 3.2'de gösterildiği gibidir.



Şekil 3.2. Döngü döşeme yöntemi ile 2-boyutlu verilerin çarpımı

T_m, T_k ve T_n döngü döşeme parametreleridir. Döşeme boyutunu ayarlamak için kullanılmaktadır. Bu şekilde hızlandırıcı, yalnızca döşeme bloklarındaki verileri depolamak için iç belleği kullanmaktadır. Girdi/Çıktı özellik haritaları ve ağırlıkları, harici bir bellekte tutulmuştur. Hızlandırıcı hesaplamaya başladığında, harici bellekten gerekli girdi özellik haritaları ve ağırlıklarını, dahili hafıza bloklarına yüklemektedir. Hızlandırıcı çıktı sonuçlarını ürettirmektedir. Bir sonraki çıktıyı hesaplamak için girdi ve ağırlık döşemeleri, girdi özellik haritalarının ve ağırlıklarının K boyutu boyunca kaymaktadır. Döngü döşeme yöntemi, GEMM hesaplaması için de uygulanmıştır. Şekil 3.3 döngü döşeme yöntemiyle evrişim hesaplamasını sunmaktadır.

Algoritma 3: Döngü döşeme kullanarak evrişim hesaplaması

Girdi: A, B
Çıktı: C
for m ← 0 **to** T_m **do**
  for k ← 0 **to** T_k **do**
    for n ← 0 **to** T_n **do**
      for t_m ← 0 **to** T_m **do**
        for t_k ← 0 **to** T_k **do**
          for t_n ← 0 **to** T_n **do**
            $C_{m+t_m,n+t_n} = A_{m+t_m,k+t_k} \times B_{k+t_k,n+t_n} + C_{m+t_m,n+t_n}$

Şekil 3.3. Evrişim döngülerinin sözde kodu

3.1.2. Döngü Açma

Bir evrişim hesaplamasını hızlandırmak için çoğu FPGA tabanlı hızlandırıcıda döngü açma kullanılmıştır (Ma ve diğ., 2018; J. Zhang ve diğ., 2017; W. Zhang ve diğ., 2020). Döngü açması, döngüdeki işlemleri birden çok bağımsız işlem olarak çoğaltarak döngü sayısını azaltmak için kullanılan bir döngü optimizasyon tekniğidir. İşlemcinin döngüdeki tüm işlemleri aynı anda gerçekleştirmesini sağlanmaktadır.

Bu tezde, U_m, U_n ve U_k, sırasıyla M, N ve K döngülerinin döngü açma parametreleridir. Açılmış döngü, döşeme döngüsünün iç döngüsü olarak düşünülebilir. Bu nedenle, açma değişkeninin değeri, döşeme değişkeninden küçük veya ona eşittir. Örneğin, $A_{m,k} \times B_{k,n \to n+U_n} + C_{m,n \to n+U_n}$ paralel olarak gerçekleştirilebilmesi için U_n ile N döngüsüne dağıtma açılmıştır. Bu şekilde, $B_{k,n}$'den $B_{k,n+U_n}$'a ve $C_{m,n}$'den $C_{m,n+U_n}$ 'a bağımsız olarak depolamak için Un boyutunda iki bellek bloğu gerekir. Ek olarak, $A_{m,k}$ 'yi tamponlanmak için Un boyutunda bir bellek bloğu gereklidir. Benzer şekilde, evrişim hesaplamalarını hızlandırmak için K ve M döngü açma yöntemi uygulanabilmektedir.

Döngü açmayı kullanmak döngüdeki tüm işlemleri aynı anda gerçekleştirildiğinden hesaplama gecikmesini büyük ölçüde azaltmaktadır. Ancak döngü açması, kaynak gereksinimlerinin sayısını doğrudan etkilemektedir. Açma parametrelerinin büyük bir değeri hem belleği hem de bilgi işlemsel kaynakları artırmaktadır. Hesaplama hızı ve kaynak kullanımı arasında bir ödünleşim vardır. Bu nedenle açma parametrelerinin en uygun konfigürasyonunu elde etmek zor bir problemdir.

3.2. Döngü Optimizasyon Analizi

Bu kısımda optimum donanım konfigürasyonunu elde etmek için döngü optimizasyon analizi tanıtılmıştır. Tasarımımızın amacı, verimli bir kaynak kullanımı ve kabul edilebilir bir işlem oranı performansı elde etmektir. Donanım konfigürasyonunu bulmak için evrişim döngülerinin hesaplama gecikmesi ve kaynak kısıtlamaları dikkate alınmaktadır. Bu çalışma, kaynak kısıtlamaları altında hesaplama gecikmesini en aza indirmeyi amaçlamaktadır.

3.2.1. Hesaplama Gecikmesi

Gecikme, çıktıyı üretmek için gereken bilgi işlem döngülerinin sayısıdır. Hesaplama gecikmesi, tüm döngülerde gerçekleştirilen toplam hesaplama döngüsü sayısından hesaplanabilmektedir. Bu nedenle döngü optimizasyon yöntemi kullanılmadığında hesaplama gecikmesi $M \times N \times K$ 'dır. Gecikme, döngü sayısını azaltmak amacıyla döngü açma yöntemi kullanılarak azaltılabilmektedir. Teorik olarak, hesaplama döngülerinin sayısı $\frac{M \times N \times K}{P_{total}}$ olmalıdır; burada P_{total} aynı anda gerçekleştirilebilen işlem sayısıdır.

Ancak evrişim döngüsü, döngü sayısını etkileyen döşenmiş ve açılmış döngüler dahil olmak üzere iç içe geçmiş döngülerden oluşmaktadır. Bu nedenle, evrişim döngüsünün hesaplama gecikmesi Eşitlik (3.1)'de gibi ifade edilebilir.

$$\#\text{Cycle}_{\text{toplam}} = \#\text{Cycle}_{\text{döşeme}} \times \#\text{Cycle}_{\text{açma}} \tag{3.1}$$

Burada $\text{Cycle}_{\text{döşeme}} = \left\lceil \frac{M}{T_m} \right\rceil \times \left\lceil \frac{N}{T_n} \right\rceil \times \left\lceil \frac{K}{T_k} \right\rceil$ ve $\text{Cycle}_{\text{açma}} = \left\lceil \frac{T_m}{U_m} \right\rceil \times \left\lceil \frac{T_n}{U_n} \right\rceil \times \left\lceil \frac{T_k}{U_k} \right\rceil$.

Döngü döşemenin hesaplama döngülerinin sayısı, T_m, T_n ve T_k parametrelerine bağlıdır. Büyük döşeme parametrelerinin kullanması, hesaplama sayısını azaltabilir. Ancak yüksek bellek kaynağı gereksinimlerine neden olur. Diğer taraftan, küçük döşeme parametreleri, yüksek sayıda hesaplama döngüsü nedeniyle uzun gecikmeye neden olmaktadır. Döngü açmasının hesaplama döngülerinin sayısı U_m, U_n ve U_k parametrelerine bağlıdır. Açma parametresinin maksimum değeri, açılmış döngülerin minimum sayıda hesaplamasıyla sonuçlanan döngü döşeme parametresine eşittir.

3.2.2. Harici Belleğe Erişim

FPGA'nın sınırlı çip üstü belleği nedeniyle, harici bellekle ilgili işlemler genellikle evrişim hesaplamaları sırasında gerçekleştirilmiştir. Harici bellek erişim işlemi, evrişim döngüsündeki diğer işlemlere kıyasla çalışma zamanını yüksek oranda tüketmiştir. Bu nedenle, optimum donanım konfigürasyonunu bulmak için bellek erişim sayısını dikkate almak gerekir. Eşitlik (3.2)'de bellek erişim hesaplama formülü verilmiştir.

$$\begin{aligned}\#\text{Bellek erişimi} &= \text{Okuma}_{\text{çıktı}} + \text{Yazma}_{\text{çıktı}} + \text{Okuma}_{\text{girdi}} \\ &\quad + \text{Okuma}_{\text{ağırlık}} \\ &= 2\left(\text{A}_{\text{çıktı}} \times \text{Buff}_{\text{çıktı}}\right) + \left(\text{A}_{\text{girdi}} \times \text{Buff}_{\text{girdi}}\right) \\ &\quad + \left(\text{A}_{\text{ağırlık}} \times \text{Buff}_{\text{ağırlık}}\right)\end{aligned} \tag{3.2}$$

Burada $\text{A}_{\text{çıktı}}$, A_{girdi} ve $\text{A}_{\text{ağırlık}}$'nin sırasıyla çıktı, girdi özellik haritaları ve ağırlıklar için kaç kez belleğe erişildiğini göstermektedir. Bellek erişim modellerinin optimizasyonu olmadan, tüm değişkenler Denklem (3.3)'teki gibi gösterilir.

$$\text{A}_{\text{çıktı}} = \text{A}_{\text{girdi}} = \text{A}_{\text{ağırlık}} = \frac{\text{M}}{\text{T}_\text{m}} \times \frac{\text{N}}{\text{T}_\text{n}} \times \frac{\text{K}}{\text{T}_\text{k}} \tag{3.3}$$

Girdi, çıktı özellik haritaları ve ağırlıklar için arabellek boyutu aşağıdaki gibi yazılır:

$$\text{Buff}_{\text{çıktı}} = \text{T}_\text{m} \times \text{T}_\text{n} \tag{3.4}$$

$$\text{Buff}_{\text{girdi}} = \text{T}_\text{k} \times \text{T}_\text{n} \tag{3.5}$$

$$\text{Buff}_{\text{ağırlık}} = \text{T}_\text{n} \times \text{T}_\text{k} \tag{3.6}$$

Minimum bellek erişimi sayısı, $\text{T}_\text{m} = \text{M}, \text{T}_\text{n} = \text{N}$ ve $\text{T}_\text{k} = \text{K}$ olduğunda elde edilmiştir. Ancak FPGA, tüm verileri bir kerede arabelleğe alacak yeterli belleğe sahip değildir. Bu nedenle, maksimum kullanılabilir arabellek boyutu atanmış ve daha sonra T_m, T_n ve T_k parametrelerini ayarlayarak en uygun konfigürasyonu bulunmuştur.

3.2.3. Kaynak Kısıtlamaları

Kaynak kısıtlamaları, ConNN'yi FPGA'ya eşlemede kritik bir zorluktur. Bu çalışmada, tüm katmanları çalıştırmak için kaynağı verimli kullanan bir tasarım elde etmek hedeflenmiştir. Kaynak gereksinimleri, döşeme ve açma değişkenlerinden tahmin edilebilmektedir. Döngü döşeme parametreleri, girdi ve çıktı özelliği haritaları ve ağırlıkları depolamak için kullanılacak belleğin boyutunu belirlemiştir. Döşeme parametreleri için en uygun konfigürasyonu bulmak zorlu bir iştir. Büyük bir arabellek kullanmak güç tüketimini artırabilmektedir. Ancak küçük bir arabellek, harici bellek erişim sayısını artırabilir. Amacımız döşeme parametreleri için uygun değerleri elde etmektedir. Optimum konfigürasyon ile kaynaklar verimli bir şekilde kullanılacak ve harici bellek erişim sayısı azaltılacaktır.

Açma parametre değerleri, evrişim yürütme sırasında gerçekleştirilen çarpanların ve toplayıcıların sayısını belirlemektedir. Açma parametre değerleri, DSP kaynak kullanımının sayısını doğrudan tanımlamıştır. 7Z020 FPGA cihazında 220 adet kullanılabilir DSP bloğu bulunur. Bu nedenle, 7Z020'deki maksimum paralel işlem sayısı, 16 bitlik çarpma için yaklaşık 400 işlemdir. Açma parametresinin değerinin FPGA'daki mevcut DSP kaynağına bağlı olduğu görülebilir. Büyük miktarda DSP'ye sahip yüksek ölçekli FPGA daha iyi verim sağlayabilir. Ancak yüksek kaynak verimliliği sağlayamamaktadır. Bu çalışmada amacımız, tasarımın kabul edilebilir işlem oranı ile verimli kaynak kullanımına ulaşması için açma parametrelerini en uygun şekilde yapılandırmaktır.

3.2.4. Konfigürasyon Analizi

Yüksek hesaplama performansı elde etmek için bellek ve DSP kısıtlamaları altında hesaplama gecikmesini ve harici bellek erişim sayısın en aza indirmek Eşitlik (3.7)'de ifade verilen kısıtlamalı bir optimizasyon problemi olarak tanımlanabilir.

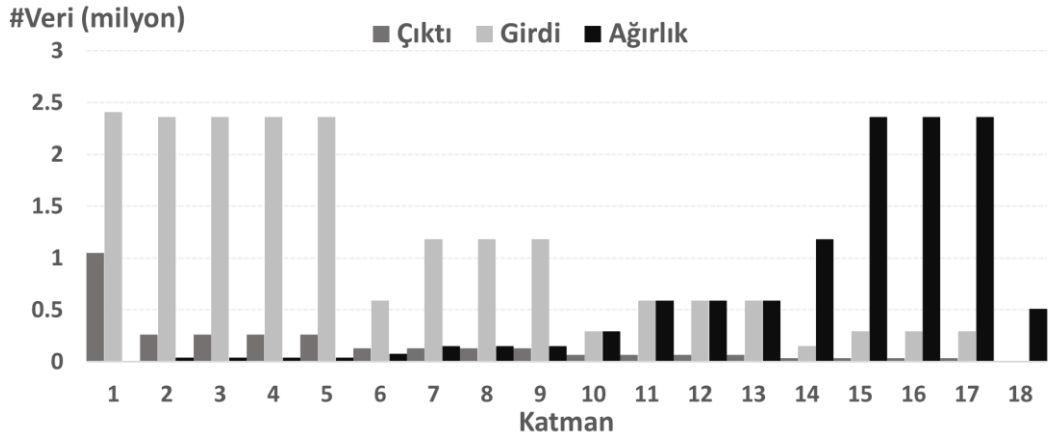
$$\min \quad \alpha \#\mathrm{Cycle}_{\mathrm{toplam}} + \beta \#\text{Bellek erişimi} \tag{3.7}$$

$$\text{kısıtlar} \quad : \mathrm{Buff}_{\mathrm{toplam}} \leq \mathrm{BRAM}_{\mathrm{maksimum}}$$
$$: \mathrm{P}_{\mathrm{toplam}} \leq \mathrm{DSP}_{\mathrm{maksimum}}$$

Burada $\#\mathrm{Cycle}_{\mathrm{toplam}} = \#\mathrm{Cycle}_{\mathrm{döşeme}} \times \#\mathrm{Cycle}_{\mathrm{açma}}$ ve paralel hesaplama sayısı: $\mathrm{P}_{\mathrm{toplam}} = \mathrm{U}_{\mathrm{m}} \times \mathrm{U}_{\mathrm{n}} \times \mathrm{U}_{\mathrm{k}}$.

Bu minimizasyon, çeşitli FPGA'larda kullanılabilir çünkü mevcut DSP ve BRAM kaynaklarındaki sınır cihaz kapasitesine göre belirlenmiştir. Paralel işlem sayısı ($\mathrm{P}_{\mathrm{toplam}}$) en büyüklendiğinde, toplam hesaplama döngüsü sayısı en aza indirilmiştir. Başka bir deyişle, hesaplama gecikmesini en aza indirmek, mevcut DSP kaynağına bağlıdır. $\mathrm{Buff}_{\mathrm{maksimum}}$ altında en uygun ortalama bellek konfigürasyonunu bulmak için veri yapısı gözlemlenmektedir. Şekil 3.4 ResNet-18 Modelindeki girdilerin, çıktı özellik haritalarının ve ağırlıklarının sayıları gösterilmiştir.

Başlangıç katmanlarındaki girdi özellik haritaların miktarı, çıkış özellik haritaları ve ağırlıklarla karşılaştırıldığında çok fazladır. Diğer taraftan, son birkaç katmanda ağırlık sayısı önemli ölçüde artmış ve özellik haritalarının sayısı azalmıştır.



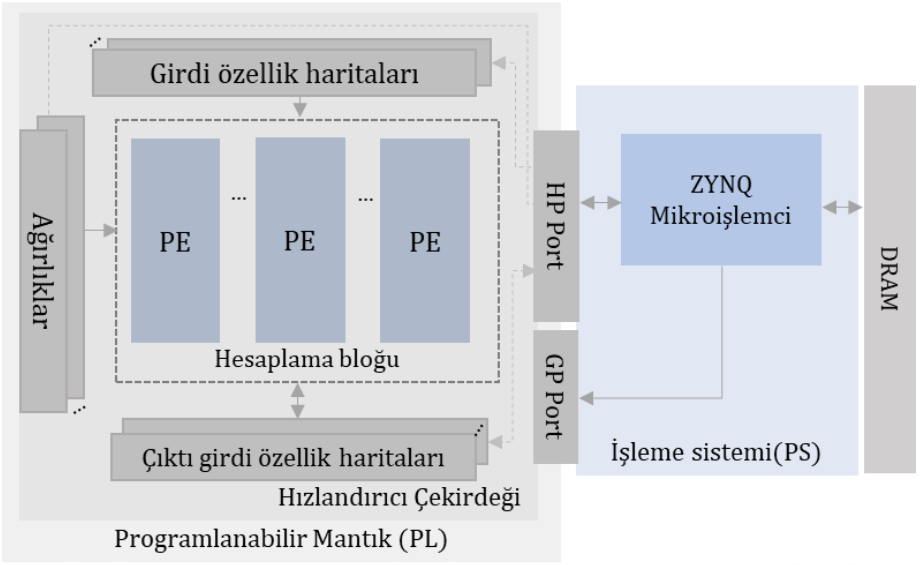
Şekil 3.4. ResNet-18'de veri yapısı

Örneğin, birinci evrişim katmanında verilerin %69,5'i girdi, %30,2'si çıktı ve %0,3'ü ağırlıklardır. Tersine, 15. evrişimsel katmanında, verilerin %11'i girdi, %1,2'si çıktı ve verilerin %87,8'i ağırlıklardır. FPGA tüm verileri depolamak için çip üzerinde bellek sağlayamamıştır. Ağırlıklar için minimum arabellek boyutunun, başlangıç katmanlarının ağırlıklarını depolamak için yeterince büyük olması gerekir. Aynı şekilde, özellik haritaları için arabellek boyutunun, son birkaç katmandaki özellik haritalarını kapsayacak kadar büyük olması gerekir çünkü bu katmanlar minimum sayıda özellik haritalarına sahiptir.

3.3. Donanım Hızlandırıcı Tasarımı

3.3.1. Donanım Hızlandırıcıya Genel Bakış

Bu kısımda, FPGA cihazlarında ConNN gerçekleştirmek için bir donanım hızlandırıcı tasarımı sunulmuştur. Tipik olarak, FPGA bir işleme sistemi (PS) ve bir programlanabilir mantık (PL) bölümünden oluşmuştur. PS bölümü, C ve C++ gibi programlama dili kullanılarak yazılım aracılığıyla programlanabilen Zynq adlı ARM tabanlı bir mikroişlemciye sahiptir. PL bölümü, sayısal devreyi çalıştırmak için programlanabilir mantık kaynakları içeren FPGA'da bir çekirdek bölümdür. Bu çalışmada, hızlandırıcı, PL bölümünde inşa edilmiş ve çalıştırılmıştır. Şekil 3.5 FPGA'da hızlandırmanın donanımla tümleşik tasarımını göstermektedir.



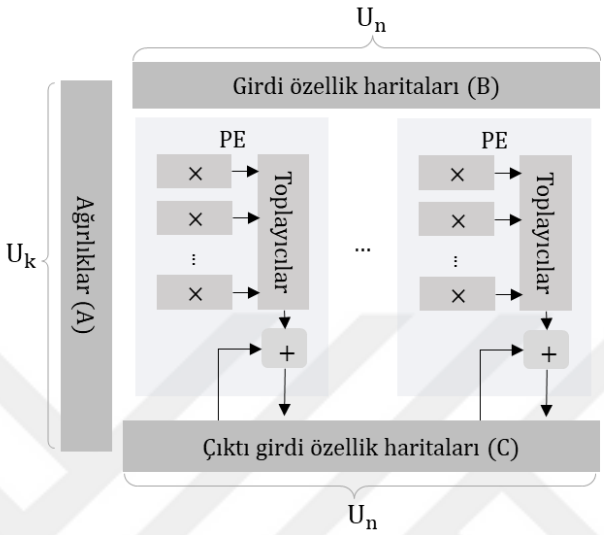
Şekil 3.5. FPGA'da ConNN için hızlandırıcının blok diyagramı

- Zynq İşlemci: Bu işlemci, gerekli çevre birimlerini başlatarak ve eğitilmiş modeli DDR kontrol modülü aracılığıyla SD karttan DRAM'e yükleyerek ConNN çıkarımının iş akışını kontrol etmektedir. Evrişimsel hesaplama gerektiğinde, Zynq işlemcisi konfigürasyon parametrelerini, girdi özelliği haritalarını ve DRAM ‘deki ağırlık adreslerini PL bölümüne geçirir. Daha sonra çıktı sonuçlarını üretmek için hızlandırıcıyı etkinleştirir.

- Hızlandırıcı Çekirdeği: Bu, sinir ağı çıkarımını çalıştırmak için donanım çekirdeğidir. Hızlandırıcı, çarpma ve biriktirme işlemleri için bir işleme elemanından (Processing Element: PE) oluşmuştur. Tek bir hızlandırıcıda birden fazla PE vardır ve tüm PE'ler paralel hesaplamalar yapmak için aynı anda çalışabilir. Bu modülün açıklamaları bir sonraki bölümde açıklanmaktadır.
- İç bellek (DRAM): Verimli veri akış ile özellik haritaları ve ağırlıklar için arabellekleri tasarlanmıştır. Her arabellek yapısı, döngü optimizasyon değişkenlerine bağlıdır. Arabellekler, eşzamanlı erişimleri desteklemek için bağımsız olarak bölümlenir.
- GP/HP bağlantı noktası: Bu, Zynq işlemci ile hızlandırıcı arasında veri göndermek ve almak için kullanılan ana arayüzdür. İşlemcinin HP bağlantı noktasını bağlamak için bellek AXI (M_AXI) kullanılır. Bu bağlantı noktası harici belleğe erişebilir. GP bağlantı, konfigürasyon parametrelerini göndermek için bir port olarak kullanılır.

3.3.2. İşleme Elemanı

Hesaplama bloğu, her bir PE'nin çarpma-biriktirme işlemi gerçekleştirdiği bir dizi işleme öğesinden oluşmuştur. Şekil 3.6 paralel hesaplama için bilgi işlem bloğu mimarisini göstermektedir.



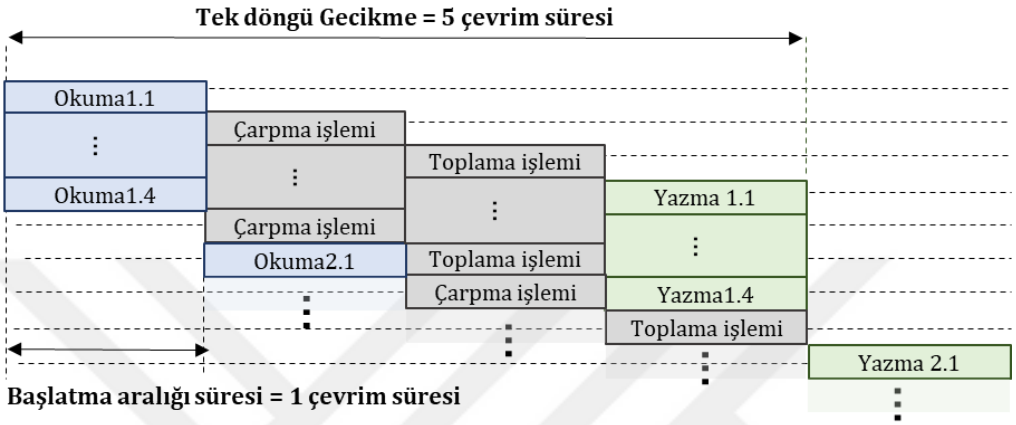
Şekil 3.6. Donanım hızlandırıcının mimarisi

PE, bir çarpan ve bir akümülatörden oluşur. Her bir PE bağımsız olarak çalışır. Böylece, bilgi işlem bloğu, evrişim hesaplamaları için çok iş parçacıklı gerçekleştirebilmektedir. PE'lerin sayısı, açma parametrelerine (U_n) bağlıdır. Her PE'nin U_k çarpanları vardır ve PE'ler farklı girdilerle çarpma-biriktirme işlemlerinden sorumludur. Ancak tüm PE'ler aynı ağırlık arabelleklerini paylaşmaktadır. Girdi arabelleği boyutu, tam bölümlere sahip $U_n \times U_k$ 'dir. Bu, tüm arabellek öğelerinin bağımsız olarak erişilebilmesini sağlar.

Ağırlık arabelleği $U_m \times U_k$ şeklindedir. Burada U_k sütunlarındaki arabellek öğeleri tam bölümlerdir. Çıktı arabelleği $U_m \times U_n$'dir. Burada U_n sütunlarındaki arabellek öğeleri tam bölümlerdir. Bu şekilde, ağırlıklarla çarpma işlemini gerçekleştirmek için girdilerin U_k elemanı PE'lere getirilir. PE'ler, çıktı arabelleğinin U_n satırlarında öğelerden verileri okur ve birikimler gerçekleştirir. Son olarak, PE'ler sonuçları arabelleğe geri yazar ve hızlandırıcı bunları harici belleğe aktarır.

3.3.3. Döngü İşlem Hattı

Başlatma aralıklarını azaltmak için hesaplama döngüsünde iş hattı tekniği kullanılır. İş hattı yöntemi kullanırken evrişimdeki işlemlerin yürütme süresi Şekil 3.7 gösterilmektedir. Bu, 4 veri elemanın evrişimi için işlem hattı yöntemini kullanmanın bir örneğidir.



Şekil 3.7. İş hattı kullanırken evrişim döngüsünün yürütme süresi

Evrişim döngüsünün çalışma zamanı 5 çevrim süresidir. Veri okumak için iki çevrim süresi, çarpma için bir çevrim süresi, toplama için bir çevrim süresi ve veri yazmak için bir çevrim süresi vardır. İşlem hattı olmadan, hesaplama döngüsünün 5 başlangıç aralığı vardır. Bu da işlemcinin bir sonraki döngüye başlamak için 5 çevrim süresi beklemesi gerektiği anlamına gelir. İşlem hattı yöntemi, tüm işlemleri aynı anda gerçekleştirerek yürütme süresini azaltabilir. Sonuç olarak, iş hattı kullanmak başlangıç aralığını bir çevrim süresine indirebilir. Bu yöntem özellikle döngü sayısı büyük olduğunda kullanışlıdır.

3.3.4. Bellek Erişim Düzeni

Bellek okuma ve yazma işlemi diğer işlemlere kıyasla yavaştır. Bellek erişim işleminin azaltılması, döngü hesaplamasının gecikmesini doğrudan azaltabilir. Bu nedenle, veriyi yeniden kullanma tekniği ile bellek erişim akışı optimize edilir. Şekil 3.8 evrişim hesaplamaları için verilerin yeniden kullanım yöntemini göstermektedir.

Algoritma 2: Evrişim döngüsü için bellek erişim modeli

Girdi: A, B
Çıktı: C
for m ← 0 **to** M **do**
  for n ← 0 **to** N **do**
    Okuma (C [m: m+T_m][n: n+ T_n])
    for k ← 0 **to** K **do**
      Okuma (B [k: k+T_k][n: n+ T_n])
      Okuma (A [k: k+T_k][m: m+ T_n])
      PE (C, B, A)
    Yazma (C [m: m+T_m][n: n+ T_n]))

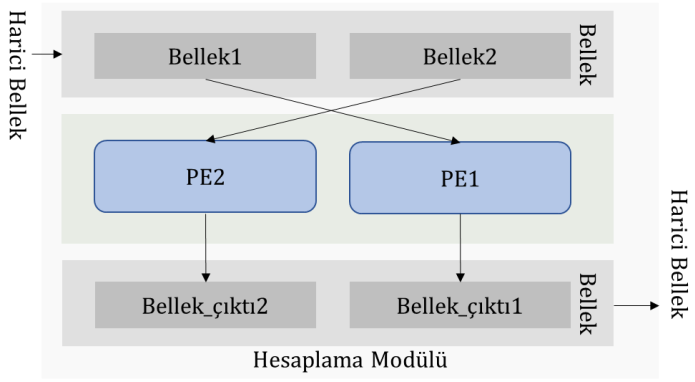
Şekil 3.8. Hızlandırıcıdaki bellek erişim düzeninin sözde kodu

Hızlandırıcı, çıktı özelliği haritalarını (C) okur ve en içteki döngüyü çalıştırmadan önce bunları arabellekte saklar. K döngüsünde, PE'lerdeki çarpma ve biriktirme işlemleri için yalnızca girdi özellik haritaları (B) ve ağırlıklar (A) arabelleklere getirilmiştir. K döngüsü tamamlandıktan sonra, hızlandırıcı, çıktıları harici belleğe geri yazmak için belleğe yazma işlemini gerçekleştirir. Bu nedenle, çıktılar için okuma/yazma işleminin sayısını K kattan fazla azalır.

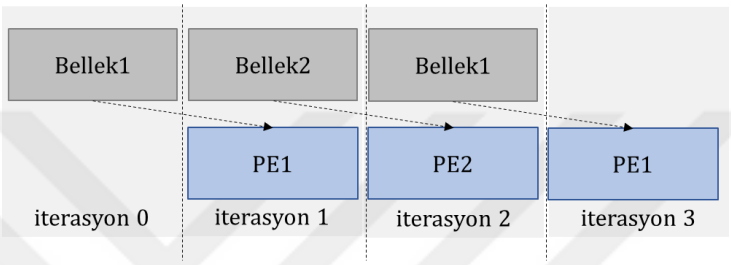
3.3.5. Örtüşen Veri Aktarımları

Örtüşen veri aktarımları (ping-pong olarak da bilinir) yöntemi, harici bellek ile hızlandırıcı arasındaki bellek bant genişliği kullanımını iyileştirmeye yönelik bir tekniktir. Her döngüde bellekle ilgili işlemler, çarpma ve biriktirme gerçekleştirmiştir. Çoklu biriktirme işlemi sırasında, bellek erişim bağlantı noktası veya bellek bant genişliği kullanılmaz. Hızlandırıcı, işlem tamamlanana kadar bekler sonra yeni verileri getirmek için bellek bağlantı noktasına erişmeye başlar.

Bu nedenle, bellek bant genişliği tam olarak kullanılmamıştır. Bu boş zaman sorununu çözmek için Şekil 3.9'de gösterildiği gibi hızlandırıcıda bir zamanlama mekanizmasına sahip iki arabellek uygulanmıştır.



(a) Çift arabelleğe alma yönteminin yapısı



(b) Örtüşen veri aktarımlarının ve bilgi işlemenin yürütme akışı

Şekil 3.9. Ping-pong yapısını kullanan çift arabellek tasarımı

Bu yöntem, verileri bellek1'e ve ardından bellek2'ye yazarak başlar. Birinci döngü, verileri bellek2'ye doldururken, PEs1, buffer1'den gelen verilerle eş zamanlı olarak evrişim hesaplamalarını gerçekleştirebilir. Benzer şekilde, PEs2, ikinci döngüde veriyi bellek1'e aktarırken, bellek2'den gelen verilerle hesaplamalar gerçekleştirir. Bu nedenle bellek erişimleri ve evrişim hesaplamaları aynı anda yapılır. Bu şekilde bellek bant genişliği kullanımı iyileştirildi.

3.4. Deneysel Sonuçlar

Bu kısımda, deneysel sonuçları paylaşmıştır. İlk olarak, açılma ve döşeme parametrelerinin nasıl yapılandırılacağını göstermek için ResNet-18 kullanılarak döngü optimizasyon sonuçları sunulmuştur. Daha sonra farklı ağ modellerini çalıştıran hızlandırıcının performansı ve hızlandırıcının kaynak kullanım sonucu verilmiştir.

3.4.1. Deneysel Kurulum

Küçük ölçekli FPGA'da ConNN hızlandırmasını sağlamak için hedef uygulama platformu olarak Zynq 7Z020 FPGA tabanlı cihaza odaklanılmıştır. Bu cihaz, 4,9 Mb çip

üstü bellek sağlamak için 140 BRAM bloğu sunar ve her biri 36Kb blok belleğe sahiptir. Ayrıca 7Z020 FPGA, 53K arama tablosuna (LUT) ve 220 DSP bloğuna sahiptir. Tüm konfigürasyonlar için 180 MHz'lik bir çalışma frekansı kullanılmıştır. Bu çalışmada hızlandırıcı çekirdekler Vivado HLS kullanılarak oluşturulmuş ve genel tasarımlar Vivado 2019.1 tarafından gerçekleştirilmiştir.

3.4.2. Performans Sonuçları

İşlem oranı, hızlandırıcının hesaplama performansını değerlendirmek için kullanılır. İşlem oranı Eşitlik (3.8)'de gibi gösterilebilir.

$$\text{İşlem oranı (GOP/s)} = \frac{\text{Toplam işlem sayısı}}{\text{İşlem zamanı (s)}} \tag{3.8}$$

Burada toplam işlem sayısı = $2 \times M \times N \times K$ dır.

İşlem oranı, gerçek dünya ortamlarında sinir ağı modellerini çalıştırmak için hızlandırıcının performansını değerlendirmede kilit faktörlerden biridir. Ancak, işlem oranı ve kaynak kullanımı ödünleşir.

Kaynak kullanımını arttırılırsa işlem oranı çok yüksek olabilir. Küçük ölçekli FPGA'lar için adil değildir çünkü küçük FPGA'lar büyük ölçekli FPGA'lara kıyasla daha az kaynağa sahiptir. Bu nedenle, tasarımın bilgi işlemsel kaynaklarını ne kadar iyi kullandığını değerlendirmek için kaynak verimliliği ölçümlerini de kullanmak gereklidir. İlgili verimlilikleri Denklem (3.9) ve Denklem (3.10)'de verilmiştir.

$$\text{DSPs verimliliği (GOP/s/DSP)} = \frac{\text{İşlem oranı (GOP/s)}}{\text{DSP sayısı}} \tag{3.9}$$

$$\text{LUTs verimliliği (GOP/s/LUT)} = \frac{\text{İşlem oranı (GOP/s)}}{\text{LUTs sayısı}} \tag{3.10}$$

Kaynak verimliliği, işlem oranının hızlandırıcı tarafından kullanılan kaynak sayısına oranıdır. Evrişim, DSP ve LUT kaynakları gerektiren çoklu biriktirme işlemi olduğundan, hızlandırıcının kaynak verimliliğini açıklamak için GOP/s/DSP ve GOP/s/LUT'leri kullanılmaktadır. Hızlandırıcının performansı Tablo 3.1'de özetlenmiştir. Görüntü sınıflandırma ve nesne algılama dahil olmak üzere farklı modeller uygulanmıştır. Her modelin yapısı ve parametre sayısı farklıdır. Sonuçlar, 7Z020 FPGA'da kullanılan kaynak

miktarını, farklı modelleri çalıştırmak için hızlandırıcının işlem oranını ve kaynak verimliliğini bildirmektedir.

Tablo 3.1. FPGA Zedboard 7Z020'de ConNN modelleri için hızlandırıcı performansı ve kaynak kullanımı

	AlexNet	ResNet-18	YOLOv3-tiny
# Parametreler	3,74M	11,5M	8,84M
# İşlemler	2,15G	4,68G	5,56G
T_m, T_n, T_k	128,768,5	256, 256, 15	255, 238, 7
U_k, U_n	5, 24	15, 8	7, 17
# LUTs	15265	13741	14215
# DSPs	129	127	128
# FFs	12641	12641	12764
En yüksek GOP/s	42,7	40,74	40,23
Ort. GOP/s	38,8	33,6	36,35
GOP/s/DSP	0,3	0,26	0,28
GOP/s/kLUT	2,54	2,45	2,56

Sonuçlar, her modelin döngü açma ve döşeme değişkenleri için farklı konfigürasyonlara sahip olduğu gösterilmiştir. İşlem oranı sonuçlarında, tüm katmanlar arasında en yüksek işlem oranı, en yüksek GOP/s'yi sunulmuştur. Hızlandırıcının model genelinde performansını açıklamak için işlem oranı ortalaması (Ort. GOP/s) da rapor edilmiştir. Sonuçlar, önerilen tasarımın AlexNet çalıştırıldığında 42,7 GOP/s'lik en yüksek işlem oranına ulaştığını göstermiştir. Kaynak verimliliği sonuçlarında, hızlandırıcı AlexNet üzerinde değerlendirme yaparken 0,3 GOP/s/DSP ile en yüksek DSP verimliliğini elde etmiştir. Hızlandırıcı, YOLOv3-tiny'yi çalıştırırken 2,56 GOP/s/kLUT'ye ulaşmıştır. Kaynak verimliliği sonuçları ayrıca, önerilen donanım tasarımı farklı modelleri çalıştırırken aynı düzeyde verimlilik sağladığından, döngü döşeme ve açıma değişkenleri için en uygun konfigürasyonu elde ettiğimizi de kanıtlamıştır.

3.5. Performans Karşılaştırması

ConNN için donanım hızlandırıcıların performans karşılaştırması Tablo 3.2'de sunulmuştur. (Rahman ve diğ., 2016), evrişim döngüsünü optimize etmek için açma döngüsü ve döşeme döngüsü yöntemleri önermişlerdir. Bu çalışma, bilgi işlem döngülerini azaltabilse de yüksek bellek erişim gereksinimleri hala yetersizdir. (J. Wang ve diğ., 2018), veri taşıma maliyetini ve harici bellek erişim gecikmesini azaltmak için

tam işlem hattı tasarımına ve ping-pong yöntemine sahip bir donanım hızlandırıcı uygulamıştır. Yazarlar ayrıca kaynak gereksinimlerini en aza indirmek için optimal bit genişliği ölçümlerini sunmuştur. (Ma ve diğ., 2018), bellek erişimlerinin gecikmesini ve maliyetini en aza indirmek için 4 seviyeli evrişim döngülerine açma döngüsü ve döşeme döngüsü yöntemleri uygulamıştır. Bu çalışma, açılma değişkenlerini ayırma çalışmamıza benzer bir stratejiye sahiptir. Bununla birlikte, çalışmaların 3-boyutlu verileri aynı anda bilgi işlemek için daha fazla döngü düzeyi vardır ve bu da bellek kaynaklarına yönelik yoğun taleple sonuçlanmıştır.

(S. E. Chang ve diğ., 2021) tarafından yakın zamanda yapılan bir çalışma, mevcut DSP'leri tam olarak kullanmak için paralel bilgi işleme ve manuel olarak tanımlanmış sayıda işleme elemanı ile birlikte döşeme döngüsü yöntemini önermiştir. Kaynak gereksinimlerini dengelemek için çarpanlar ve bit kaydırıcılar kullanılarak iki farklı bilgi işlem çekirdeği uygulanmıştır. Çalışma zamanı dizini eşleşmesini önlemek için (L. Lu ve diğ., 2019), ağırlıkların ve gerekli girdi piksellerinin eşlenmesi için arama tablosu içeren bir döşeme döngüsü önermiştir. Bunun yanında, paralel hesaplama ve sürekli ağırlık depolama sağlayan veri akışının dahili hesaplamasına sahip bir iş hattı sunulmuştur. (Guo ve diğ., 2017) çekirdek verilerine, girdi ve çıktı piksellerine 3 seviyeli paralellik uygulamıştır. Ek olarak, yazarlar, çip üzerindeki önbelleği tam olarak kullanmak için toplam bellek gereksinimi ve veri akışı optimizasyonunun bir analizini sunmuştur. (T. Y. Lu ve diğ., 2020), küçük ölçekli FPGA cihazları için bilgi işlem talebini ve donanım kaynaklarını azaltmak adına nicemleme yöntemlerini araştırmışlardır. Bu da minimum bilgi işlem ve depolama kaynaklarıyla sonuçlanmıştır. Bununla birlikte, donanım optimizasyonu araştırılmadığından, yaklaşımları diğer çalışmalara kıyasla en düşük işlem oranını vermiştir.

Özetle, işlem oranı performansı FPGA'nın mevcut kaynakları ile ilgilidir. Yüksek performanslı FPGA tabanlı uygulamalar, FPGA'da kullanılanlara kıyasla yüksek işlem oranı performansı sağlayabilmiştir. Ancak amacımız, karmaşık bilgi işlemsel modellerini sınırlı kaynaklara sahip küçük ölçekli FPGA'ya eşlemektir. Bu nedenle, kaynak verimliliği ölçütleri, kaynaklar sınırlı olduğunda tasarımın ne kadar iyi performans gösterdiğini ölçmede birincil faktördür. Sonuçlar, önerilen hızlandırıcının diğer FPGA tabanlı hızlandırıcılarla karşılaştırılabilir kaynak verimliliği sağladığını göstermiştir.

Tablo 3.2. ConNN'nin önceki FPGA tabanlı hızlandırması ile performans karşılaştırması

Yöntem	Model	Cihaz	Bit	# LUTs	#DSP	# BRAM	# FFs	GOP/s	GOP/s/DSP	GOP/s/kLUT
(Rahman ve diğ., 2016)	AlexNet	VC709	32	45K	2688	543 (36Kb)	-	147,8	0,05	3,28
(J. Wang ve diğ., 2018)		ZC706	8	86282	808	300	51387	493	0,61	5,47
(Ma ve diğ., 2018)		Stratix-V	8,16	121K	256	1552	-	134,1	0,52	1,1
Önerilen yöntem		7Z020	16	15265	129	218	12641	38,8	0,3	2,54
(L. Lu ve diğ., 2019)	ResNet-18	ZCU102	16	552K	1144	912	-	291	0,25	0,53
(Chang ve diğ., 2021)		7Z020	4	28K	220	112	17083	77	0,35	2,72
Önerilen yöntem		7Z020	16	13741	127	174	12287	33,6	0,26	2,45
(Guo ve diğ., 2017)	YOLO model tabanı	7Z020	8	29867	190	85,5	35489	62,9	0,33	2,17
(T.Y. Lu ve diğ., 2020)		7Z020	8, 4	49158	124	104	-	26,73	0,22	0,54
(Chang ve diğ., 2021)		7Z020	4	28288	220	56	17083	84	0,38	2,97
Önerilen yöntem		7Z020	16	14215	128	166	12764	36,35	0,28	2,56

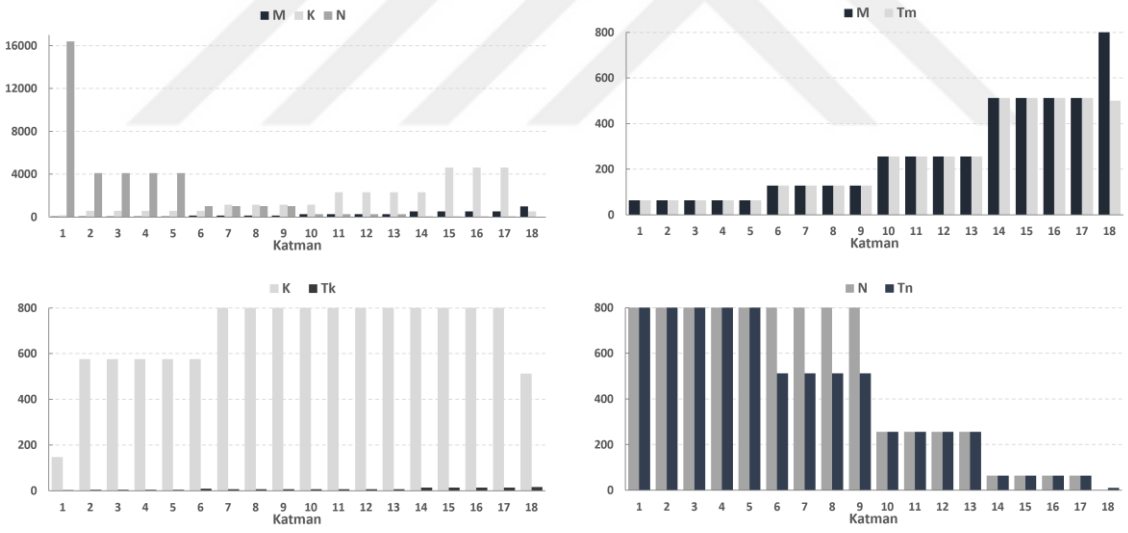
3.6. Sonuçlar

Bu bölümde, döngü optimizasyon teknikleri kullanarak evrişimsel sinir ağının hızlandırılması sunulmuştur. Açma döngüsü ve döşeme döngüsü analizi, mevcut kaynaklarla ilgili en uygun konfigürasyonun araştırılmasıyla birlikte sunulmuştur. Donanım bölümünde, hızlandırıcının mimarisi ve veri akışı optimizasyon yöntemleri önerilmiştir. Önerilen hızlandırıcının performansı ve kaynak verimliliği detaylı bir şekilde irdelenmiştir. ResNet, AlexNet ve YOLOv3-tiny dahil olmak üzere ConNN modelleri önerilen çalışmayla eşleştirilmiştir. Deneysel sonuçlar, önerilen hızlandırıcının 42,7 GOP/s'lik en yüksek işlem oranı elde ettiğini göstermektedir. Ayrıca tasarımımız 0,3 GOP/s/DSP ve 2,56 GOP/s/kLUT kaynak verimliliği sağlayabilmektedir.

4. KAYNAK VERİMLİ UYARLANABİLİR HIZLANDIRICI

4.1. Tanıtım

Önceki bölümde, statik donanım kullanan FPGA üzerinde ConNN hızlandırıcısı anlatılmıştır. Statik hızlandırıcının avantajı, farklı ConNN modelleriyle uyumlu olması ve uygulamada basit olmasıdır. Ancak ConNN modelindeki katmanlar, hesaplama iş yükü ve değişken sayısı açısından farklı özelliklere sahiptir. Dolayısıyla tüm katmanlar için belirli bir donanım yapılandırmasının kullanılması yalnızca en iyi ortalama performansı sağlayabilir. Katman yapılarının çeşitliliğini gözlemlemek için döngü optimizasyon yöntemi her katmanda ayrı ayrı uygulanır. ResNet-18 modelinde her katman için optimizasyon sonrası döşeme değişkenlerinin değerleri Şekil 4.1'de gösterilmiştir. Sol üstteki grafik, döngü parametrelerinin değerlerini verir ve şeklin geri kalanı, döngü parametreleriyle karşılaştırıldığında döngü döşeme parametrelerinin değerlerini gösterir.



Şekil 4.1. Döşeme parametreleri ile gerçek döngü değerleri arasında bir karşılaştırma

Her katman için döşeme değişkenlerinin optimal konfigürasyonu farklıdır. T_m'nin değeri, önceki katmanlarda küçüktür ve daha derin katmanlarda artar. Tersine, T_n'nin değeri başlangıçta büyüktür ve daha derin katmanlarda azalma eğilimindedir. T_k'nin değeri göreceli olarak artar, ancak K parametresine kıyasla çok küçüktür.

Statik hızlandırıcının uygulanması için ConNN modelinde tüm katmanları çalıştırırken hızlandırıcının performansı göz önüne alınarak optimal döşeme ve açma parametreleri elde edilmiştir. Bu nedenle, döşeme ve açma parametreleri, hızlandırıcıdaki kullanılabilir arabelleğin boyutunu ve işleme öğelerinin boyutlarını gösterir. Dolayısıyla hızlandırıcının tüm katmanları gerçekleştirmesi için aynı donanım yapılandırması kullanılır. Bununla birlikte, statik olarak yapılandırılmış hızlandırıcı, yukarıdaki gözlemimiz gibi katman yapısının çeşitliliği nedeniyle standardın altında performansa yol açar. Sorunu açıklamak için $T_n = 256$ ve $T_k = 8$ olsun. Nitekim, girdiler için arabellek boyutu 2048'dir. Sabit donanım yapılandırmasının kullanılmasından kaynaklanan sorunlar aşağıda sunulmuştur.

- Kaynakların yetersiz kullanımı: ResNet-18'de çalıştırmak için yukarıdaki yapılandırmayı kullanarak, girdi verilerinin boyutu arabellek boyutundan daha büyük olduğundan, katman 1'den katman 13'e kadar girdi arabelleği tam olarak kullanılır. Ancak katman 14'ten katman 18'e kadar girdi verilerinin miktarı, kullanılabilir arabelleğin yalnızca %25'i kadardır. Bu nedenle, arabelleğin %75'i kullanılabilir durumdadır ama kullanımda değildir. Ayrıca, döngü parametresi döşeme değişkeninden büyük olduğunda ancak döşeme değişkenine bölünemediğinden yetersiz kullanım meydana gelir. Böylece girdilerin son bölümündeki veri sayısı arabellek boyutundan daha azdır. Girdilerin son kısmı arabelleğe alındığında arabellek tam olarak kullanılmaz. Her iki senaryo da verimsiz bellek kullanımına neden olur.

- Hesaplamaların dengesizliği: Hızlandırıcı, iki boyutlu bir paralel hesaplama modülü olan bir dizi işleme öğesinden oluşur. Hesaplama modülünün boyutları, U_n ve U_k açma parametreleri tarafından belirlenmiştir. N ve K döngü parametrelerinin değerleri önemli ölçüde değiştirilirse, sabit hesaplama modülü uygun paralellik boyutlarını sağlayamaz. Örneğin, ResNet-18'in 2. katmanında, döngü parametreleri N ve K sırasıyla 4096 ve 64'tür. N boyutundaki iş yükü ağır olduğu için açma değişkenini U_k'den daha yüksektir. Ancak bu katman 14 için verimsiz bir konfigürasyon olacaktır çünkü bu katmanda N ve K değerleri sırasıyla 64 ve 512'dir. Sonuç olarak, hesaplama kaynakları ResNet-18'deki tüm katmanlar için etkin bir şekilde kullanılmamaktadır.

Yukarıdaki sorunları çözmek için hızlandırıcının mimarisi evrişim katman özelliklerine göre ayarlanabilir olmalıdır. Hızlandırıcıdaki bilgi işlem modülleri ve bellek yapısı, çalışma zamanı sırasında güncellenebilir. Bu nedenle, bu bölümde, katman yapısının çeşitliliği nedeniyle farklı ConNN hesaplamalarını kolaylaştırmak için dinamik olarak yeniden yapılandırılabilir bir donanım hızlandırıcı sunulmuştur.

4.2. İlgili Çalışmalar

ConNN hızlandırma üzerindeki çalışmaların çoğu, ConNN ‘deki tüm evrişimsel katmanlarını çalıştırmak için belirli bir donanım hızlandırıcısı uygulamıştır. Ancak farklı katmanlar farklı yapısal özelliklere sahip olduğundan, statik tasarımın optimumun altında olmasına neden olabilir. Katmanların çeşitliliği sorununu ele almak için donanım uyarlamalı hızlandırıcıları araştıran az sayıda çalışma vardır. ConNN modellerinin eğitimi için FPGA tabanlı çerçeve (W. Zhao ve diğ., 2016) içinde önerilmiştir. Önerilen algoritmalar, bant genişliği ve donanım kaynağı kısıtlamaları altında performansı en üst düzeye çıkarmak için her katmandaki donanım yapılandırmasını optimize eder.

Hesaplama modülleri, ConNN eğitiminde çalışma zamanı sırasında alttan üst katmana sırayla yapılandırılır. İleriye dönük hesaplama için çerçeve, bilgi işlem modülünü üstten alt katmana yeniden yapılandırabilir. Yazarlar (Putic ve diğ., 2018), hızlandırıcı parametrelerinin katmanın özelliklerine göre dinamik olarak yeniden yapılandırılabileceği dinamik donanım yeniden yapılandırması önermişlerdir. Hızlandırıcının mimarisi, her katman için bilgi işlem öğelerinin sayısı ve bellek kapasitesi en uygun şekilde yapılandırılarak tasarlanmıştır. Hızlandırıcının donanımı, en uygun statik yapılandırmaya kıyasla yeni yapılandırmanın çıktı performansına dayalı olarak sırayla güncellenir. (Kästner ve diğ., 2018)'deki bir çalışma, sinir ağının dinamik yeniden yapılandırılması ile FPGA'lara eşlenmesi için bir tasarım iş akışı önermiştir. Yazarlar, her katman için en uygun veri akışını ayrı ayrı araştırmıştır. Donanım hızlandırmayı çalıştırmak için yalnızca bir yeniden yapılandırılabilir bölüm oluşturulur. Katmanlar, veri akışı gereksinimine göre konfigürasyonlara girer ve çıkar. Dinamik kısmi yeniden yapılandırmaya sahip hızlanan bir ikili eşleştirilmiş ConNN (Skrimponis ve diğ., 2020) içinde tanıtılmıştır. ConNN modelindeki katmanlar, veri kesinliğine ve katman özelliklerine göre gruplara ayrılmıştır. Her katman grubu için veri akışlarının sıralanmasını içeren bir donanım hızlandırıcı çekirdeği tasarlanmıştır.

Uyarlanabilir ve hiyerarşik ConNN yöntemi (Farhadi ve diğ., 2019) içinde uygulanmıştır. Ağ modelleri, hafif bir ConNN, bir karar katmanı ve derin kısım olarak adlandırılan karmaşık bir ConNN ‘den oluşur. Karar katmanı, önceki katmanın performansını değerlendirir ve ardından ConNN modeli, karar katmanının çıktısına göre mimarisini değiştirebilir. Her katman için donanım konfigürasyonunu optimize etmek yerine, bu çalışma (Gong ve diğ., 2020) kümeleme algoritması ile ağ haritalama yöntemini geliştirmiştir. Her kümede katmanları çalıştırmak için FPGA yer planını birden çok kez yeniden yapılandırabilir bölüme bölmüşlerdir. Hızlandırıcı, bilgi işlem modüllerinin sayısını, arabellek boyutunu ve veri akışı modellerini dinamik olarak güncelleyebilir.

Bu çalışmada, önerilen hızlandırıcı, arabellek boyutları ve hesaplama elemanlarının sayısı dahil olmak üzere donanım yapısını dinamik olarak değiştirebilir. Ağ kümeleme, katmanların yapısına göre ağı verimli bir şekilde gruplandırmak için uygulanır. Dinamik hızlandırıcımızın genel amacı, tatmin edici performansı korurken mümkün olan en yüksek kaynak verimliliğini elde etmektir.

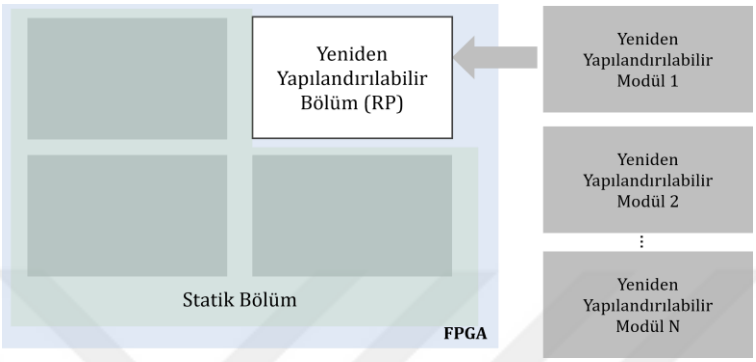
4.3. Dinamik Kısmi Yeniden Yapılandırma Yöntemi

4.3.1. Tanıtım

Genelde FPGA'lar, donanım modülünü farklı bir donanım konfigürasyonu ile yeniden programlayarak özelleştirme yeteneğine sahiptir. Yeniden yapılandırma sırasında, FPGA'lar tüm donanım konfigürasyonlarını temizlemek için sıfırlama moduna geçer ve daha sonra tüm mantık değiştirilir. Bu yapılandırma şeması, her konfigürasyonun ayrı bir veri akışı tasarımında uygulandığı hızlandırıcı ile kullanılabilir. Ancak bunu yaparken, hızlandırıcının durması gerekir ve yeni tam bit akışı dosyası yapılandırma belleğine yüklenebilir. Bu şekilde, tasarımın ConNN çıkarımını yürütmesi gerekenden daha fazla zaman alır.

FPGA'lar için daha fazla donanım esnekliği sağlayan kısmi yeniden yapılandırma (Partial reconfiguration: PR) adı verilen gelişmiş bir yeniden yapılandırma tekniği vardır. PR, farklı alanlardaki modüller aktif kalırken tasarımcıların belirli bir alandaki donanım modülünü yeniden programlamalarına olanak tanır. Bu şema, yeniden yapılandırma işlemi sırasında FPGA'nın sıfırlama durumunda olmasını gerektirmez. Bu nedenle,

hızlandırıcı için PR kullanmak, FPGA cihazlarında oldukça esnek yapılandırma ile dinamik donanım işlevlerini etkinleştirebilir. PR kavramında, FPGA'daki alan iki bölüme ayrılır: statik bölüm (Static partition: SP) ve yeniden yapılandırılabilir bölüm (Reconfigurable partition: RP). Şekil 4.2'de yeniden yapılandırılabilir tek bir bölümle PR bölgesi kavramı gösterilmektedir.



Şekil 4.2. FPGA'da kısmi yeniden yapılandırma mimarisine genel bakış

RP kısıtlı bir karedir ve tek bir alan veya her birinin bağımsız olarak çalışabileceği birden fazla alan olabilir. Arama tablosu (LUT), flip-flop (FF), blok belleği (BRAM) ve sayısal işaret işleme (DSP) blokları gibi ortak kaynaklar RP'de mevcuttur. SP'deki donanım modülü, yapılandırma bağlantı noktaları olarak atanan mantık hücrelerini kullanarak RP'deki donanımla iletişim kurabilir. Bu bağlantı portları aynı zamanda referans noktaları olarak da çalışır. Tasarımcılar, RP'de çalıştırmak için farklı donanım blokları uygulayabilir, ancak tüm donanım bloklarının aynı bağlantı portunu kullanması gerekir. Bu nedenle, RP ve SP arasındaki iletişim portunun yeniden yapılandırılmasına gerek yoktur. PR kullanmanın avantajları aşağıdaki gibi özetlenebilir.

- PR, aynı kaynaklar farklı donanım modülleri için kullanılabildiğinden artan bir mantık kapasitesi sağlar. Bu özellik daha büyük devrelerin sınırlı kaynak FPGA cihazlarına yerleştirilmesine izin verir.

- PR ile donanım uygulaması, kaynak kullanımını ve güç tüketimini daha da iyileştirebilir. Donanım modülü devre dışıyken veya kullanılmadığında, tasarımcılar kullanılmayan donanım modüllerini boş alanlarla değiştirebilir veya yakında kullanılacak mantık modülleriyle değiştirebilir.

- FPGA'larda PR ile mantık devrelerini yeniden oluşturmak, yapılandırma süresini azaltır. Kısmı bit akışı dosyasının boyutu, tam bit akışı (.bit) dosyasına kıyasla küçüktür. Sonuç olarak, PR kullanan yapılandırma süresi, tam yeniden yapılandırmadan önemli ölçüde daha azdır.

Son zamanlarda, PR, Xilinx'in tüm yeni FPGA'larında ve bazılarında Intel tarafından desteklenmektedir. Xilinx da PR'yı kullanıcılar için daha kolay hale getirmek adına Dynamic function eXchange (DFX) adlı yeni bir tasarım aracı geliştirilmiştir. Ancak PR uygulaması hala karmaşık bir süreçtir ve geleneksel FPGA tasarımından farklıdır. FPGA mimarisinde yüksek deneyim ve bilgi gerektirir.

4.3.2. Yeniden Yapılandırma Kontrolörü

Yeniden yapılandırma süresi, özellikle yürütme süresinin önemli olduğu bir sistemde sistemin genel performansını etkilediği için önemli bir faktördür. Yeniden yapılandırma kontrolörü, yeniden yapılandırma süresinde önemli bir rol oynar. Xilinx 'in FPGA'ları aşağıdaki gibi iki yeniden yapılandırma kontrolörü sağlar.

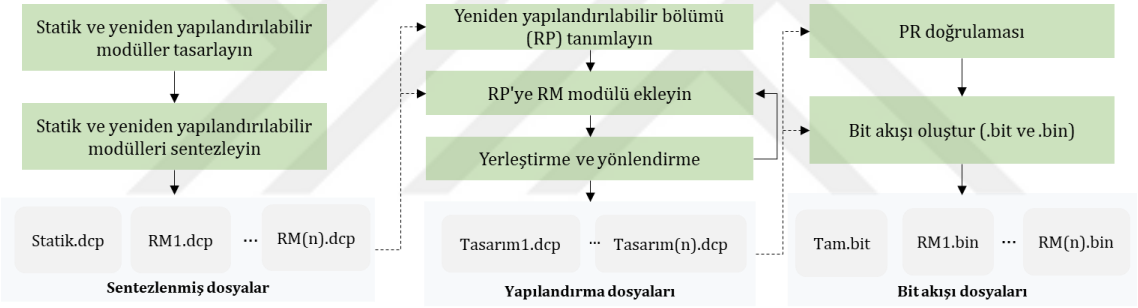
- Dahili yapılandırma erişim portu (ICAP) kontrolörü: Bu, PL bölümünde sağlanan yapılandırma denetleyicisidir. ICAP, bir bilgisayardan FPGA'ları programlamak için kullanılan harici bir konfigürasyon arayüzü olan SelectMAP ile aynı şekilde çalışır. Xilinx ayrıca AXI ara bağlantı portunu kullanarak PS bölümünde bulunan işlemciden ICAP'yi çalıştırmak için HWICAP IP adlı bir kontrol modülü sunar. FPGA'ları kısmi bit akışı dosyası ile yeniden yapılandırmak için HWICAP kullanmak, 67 MByte/s'lik bir verim elde edebilir.

- İşlemci yapılandırması erişim portu (PCAP) kontrolörü: FPGA cihazlarının PS bölümünde bulunan başka bir kontrolördür. Tasarımcılar, PS'deki yapılandırma kontrolörü aracılığıyla PL bölümündeki mantığı yeniden yapılandırabilir. Bu kontrolör, bit akışı dosyalarını harici bellekten PCAP'ye aktarmak için bir DMA denetleyicisi olan cihaz yapılandırma arayüzüne (DevC) bağlanır. Bu modül PS bölümünde olduğu için PL bölümünden kaynaklara ihtiyaç duyulmaz. Ancak PCAP ile yeniden yapılandırma, Zynq işlemcisinin iş yüklerini artırır. PCAP modülü, 128 MByte/sn'lik yeniden yapılandırma verimine ulaşabilir.

Yukarıdaki yapılandırma modülleri, ConNN'in hızlandırılması için hala düşük işlem oranı sunmaktadır. Bunu geliştirmek için bu tezde yüksek verimli açık kaynaklı bir yapılandırma kontrolörü olan ZyCAP'yi (Vipin ve diğ., 2014) benimsenmiştir. ZyCAP, 808 FF ve 620 LUT tüketen ICAP, DMA denetleyicisi ve ICAP yöneticisinden oluşur. Bu kontrolör, PCAP ve ICAP'den sırasıyla 2× ve 5,7× daha hızlı olan 382 MByte/s'lik verime ulaşır. Tasarımımızda bu modül statik alandaki modüllerden biri olarak entegre edilmiş ve yeniden yapılandırma işlemi için kullanılmıştır.

4.3.3. Süreç Tasarımı

PR kullanarak donanım uygulama süreci, FPGA'ların geleneksel uygulamasından farklıdır. Tam ve kısmi bit akışı dahil olmak üzere konfigürasyon dosyalarını oluşturmak için ek işlemler gereklidir. Şekil 4.3'de PR kullanırken donanım tasarımı süreci gösterilmektedir.



Şekil 4.3. Kısmi yeniden yapılandırma özelliği kullanılarak FPGA için donanım modüllerinin süreç tasarımı

İlk adım, statik ve yeniden yapılandırılabilir bir modül oluşturmaktır. Statik ve yeniden yapılandırılabilir parçalar için donanım modüllerini ayrı ayrı uygulayabilir. RP'de dağıtılacak mantığa yeniden yapılandırabilir modülü (Reconfigurable module: RM) denir. Donanım tasarımını tamamladıktan sonra çıktı dosyasını bir tasarım kontrol noktası (Design checkpoint: dcp) olarak oluşturmak için sentez sürecini başlatabilir. Statik modüller tek bir entegre tasarım olarak sentezlenirken RM'ler ayrı ayrı sentezlenir. Bu nedenle, sentez işleminden sonra RM'lerin bir statik dcp ve birkaç dcp dosyası olur. Bir sonraki adım, FPGA kat planından yeniden yapılandırılabilir bölümü tanımlamaktır. Yeniden yapılandırılabilir bölüm, tasarımcının boyutlarını belirlediği kare bir alandır. Bu kısıtlama alanı, RM'leri dağıtmak için kaynaklar içerir. Sonraki işlem, RM'nin

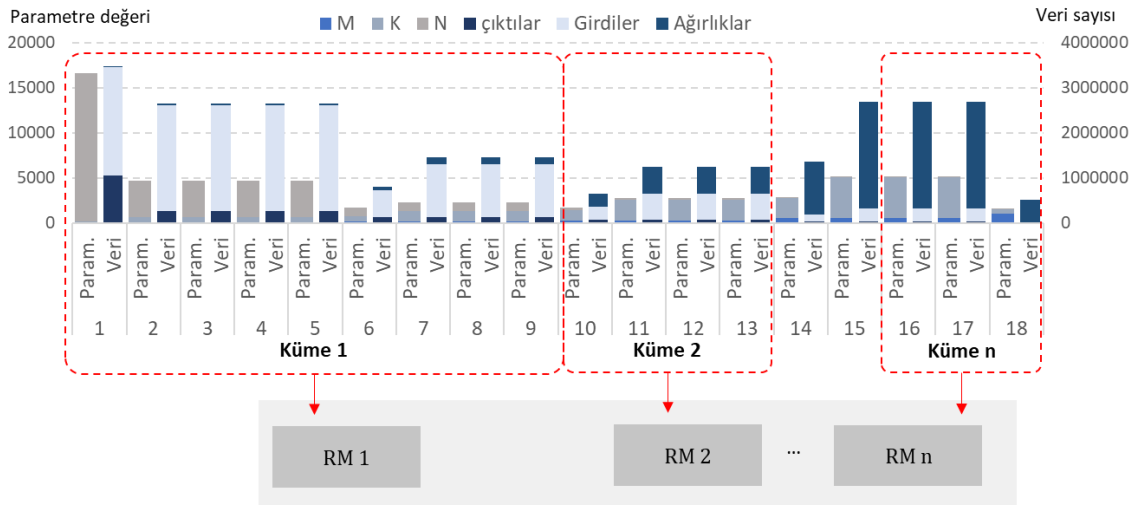
sentezlenmiş dosyasını (.dcp) yeniden yapılandırılabilir bölüme yüklemektir. Tek bir bölümde yalnızca bir RM kullanılabilir. Daha sonra hem statik hem de yeniden yapılandırılabilir bölgelerdeki donanım modülleri kablo ağ listesini oluşturmak için yerleştirme ve yönlendirme işlemleri gerçekleştirilir.

Donanım modülünün net listesi bir dcp dosyası olarak kaydedilir. İlk RM için yerleştirme ve yönlendirme işlemlerini tamamladıktan sonra statik bölgede netlist kilitlenip RP'de netlist kaldırılır. Daha sonra bir sonraki sentezlenen RM, RP'ye yüklenir ve bu RM ağ listesini oluşturmak için yer ve rota işlemleri tekrar yapılır. Bu işlem tüm RM'ler için tekrarlanır. Son olarak, bit akışı oluşturma işlemi, tam ve kısmi bit akışı dosyaları dahil olmak üzere bit dosyasını oluşturmak için ait olduğu yerdeki netlist dosyasını kullanır ve süreçleri yönlendirir.

4.4. Uyarlanabilir Yeniden Yapılandırılabilir Hızlandırıcı

4.4.1. Genel Bakış

Katman yapılarının çeşitliliğini kolaylaştırmak için ConNN çıkarımı sırasında farklı donanım konfigürasyonlarıyla çalışabilen uyarlanabilir bir hızlandırıcı sunulmuştur. Şekil 4.4'te PR kullanan FPGA'da uyarlanabilir hızlandırıcı ile ConNN uygulaması kavramı gösterilmektedir.



Şekil 4.4. Uyarlanabilir hızlandırıcı ve katman kümeleme yöntemini kullanarak FPGA üzerinde ConNN uygulamasına genel bakış

ConNN modeli çok sayıda katmandan oluşur. Dolayısıyla, her katman için donanım tasarımı maliyetli ve zaman alıcıdır. Benzer yapıya sahip katmanların, aynı donanım yapılandırmasıyla en iyi performansı elde etme eğiliminde olduğu fark edilmiştir. Bu nedenle, donanım yapısını tek tek katman için ayarlamak yerine, donanım uygulama tasarımlarının sayısını azaltmak için önce katmanlar yapısal özelliklerine göre bir küme halinde sınıflandırılmıştır. Daha sonra, her kümedeki katmanları barındırmak için donanım yapısı optimize edilmiştir.

FPGA, tüm yapılandırmaları bir kerede uygulamak için yeterli olmayan sınırlı kaynaklara sahiptir. Bu nedenle, PR, yalnızca gerekli donanım modülünü yerleştirerek ve aynı kaynakları kullanan diğerleriyle değiştirerek küçük ölçekli FPGA cihazlarında çoklu donanım mantık modüllerinin dağıtımını sağlamak için benimsenmiştir. Bu nedenle hızlandırıcı, ConNN çıkarımı sırasında yapısını katmanın özelliklerine göre dinamik olarak ayarlar.

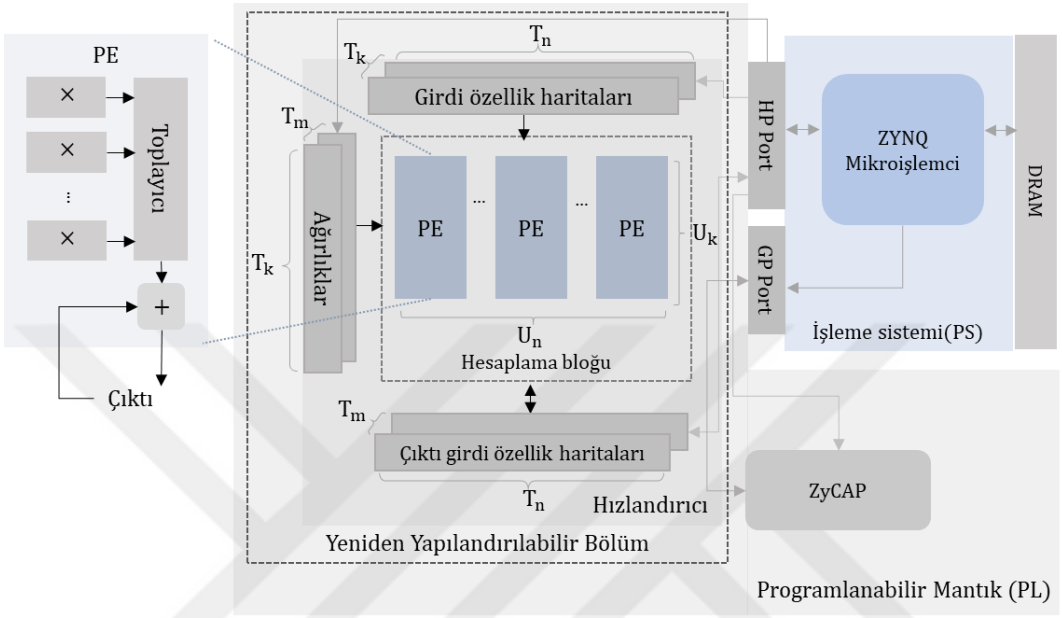
PR ile ConNN hızlandırmayı uygulamak için FPGA'daki mantık alanı manuel olarak statik ve yeniden yapılandırılabilir bölümlere ayrılır. Hızlandırıcı, yeniden yapılandırma bölümünde uygulanır ve donanım yapılandırması, yeniden yapılandırılabilir bir modül (RM) olarak tasarlanmıştır. Hızlandırıcı ConNN çıkarımı gerçekleştirdiğinde, diğer mantık modülleri hala çalışırken RP'deki RM'yi yeniden yapılandıracaktır.

4.4.2. Hızlandırıcı Mimarisi

FPGA üzerinde uyarlanabilir hızlandırıcının mimarisi Şekil 4.5'te gösterilmektedir. Donanım yapısı, bellek arabelleklerinden ve evrişim hesaplaması için hesaplama bloğundan oluşan statik tasarımla aynıdır. Hızlandırıcı, işleme sistemi (PS) Zynq işlemcisi ile Zynq GP bağlantı noktası üzerinden iletişim kurabilir. HP bağlantı noktası, harici bellek ile hızlandırıcı arasında veri akışı için kullanılır. Uyarlanabilir tasarımda, bellek bloğunun ve bilgi işlem bloğunun boyutları çalışma zamanı sırasında değiştirilebilir.

ConNN çıkarımı sırasında hızlandırıcının bağlantı portları değişmediği için statik bölmede HP ve GP portlarının arayüz modülü uygulanır. ZyCAP, yeniden yapılandırma sürecinin denetleyicisidir. Bu modül statik bölümde uygulanmıştır. Dinamik donanım

özelliklerini etkinleştirmek için hızlandırıcı, tüm programlanabilir mantık alanından bölünmüş bir alan yeniden yapılandırılabilir bölüme yerleştirilmiştir. Yeniden yapılandırma bölümü olarak yalnızca bir bölge tahsis edilir. Tasarımcılar aynı sınırlarla kaynak kullanımını optimize edebildiğinden, bu basit ama etkili bir yöntemdir.



Şekil 4.5. Uyarlanabilir hızlandırıcının genel bakış mimarisi

Hızlandırıcı, T_m, T_n ve T_k değişkenlerini ayrı ayrı yapılandırarak dinamik arabellek boyutları sağlar. Burada $T_n \times T_k$, $T_n \times T_m$ ve $T_k \times T_m$ sırasıyla girdilerin, çıktıların ve ağırlık arabelleklerinin boyutlarıdır. Dinamik arabellekler, katmanlardaki veriler için en uygun arabellek boyutunu ayarlayarak bellek kaynaklarının yetersiz kullanılması sorununu çözer. Ayrıca, yüksek sayıda bellek erişiminin neden olduğu genel gecikmeyi de azaltır. Bununla birlikte, ayarlanabilir arabellek boyutu, işleme elemanlarının yeterli bir oranda bir çıktı üretememesi durumunda oluşabilecek hesaplama aşırı yüklenmesini önlemek için karşılık gelen hesaplama modüllerini gerektirir. Bu nedenle, işlem elemanlarının sayısı, U_n ve U_k değişkenleri düzenlenerek ayarlanabilir.

Evrişim hesaplamaları süreci, hızlandırıcının ilk katman için en uygun konfigürasyonu kullanarak çalıştırılmasıyla başlar. Aynı zamanda, bir sonraki donanım konfigürasyonu DDR belleğe yüklenir. Girdi, çıktı ve ağırlık verileri, HP bağlantı noktası aracılığıyla harici bellekten alınır. M, N ve K döngü parametreleri, GP bağlantı noktası kullanılarak Zynq işlemcisinden hızlandırıcıya gönderilir. Hızlandırıcı daha sonra ConNN çıkarımını

gerçekleştirir ve çıkarımı tamamladıktan sonra sonuçları Zynq'e geri gönderir. ConNN çıkarımı sırasında, açma ve döşeme değişkenlerinin güncellenmesi gerekiyorsa, ZyCAP yüklenen yapılandırmayı okuyarak ve program belleğini güncelleyerek yeniden yapılandırma sürecine geçer. Yeniden yapılandırma işlemini tamamladıktan sonra, hızlandırıcı veri alma işlemini tekrar eder ve hızlandırıcının yeniden yapılandırma için durdurulduğu katmandan çıkarım yapar.

4.4.3. Ağ Bölümleme

Kaynakların yetersiz kullanımını ele almak için her katmanın döngü değişkenlerinin optimal konfigürasyonunu bulmak çözüm olabilir. Ancak ConNN'nin mimarisi, farklı yapılara sahip yığılmış katmanlardan oluşur. Ayrıca karmaşık bir ConNN modelinde katman sayısı 100'den fazla olabilir. Her katman için donanım tasarlamak zaman alıcı ve maliyetlidir. Ayrıca, kaynak kısıtlamaları nedeniyle tüm yapılandırmalar FPGA'da aynı anda dağıtılamaz. Üstelik, katman sayısı büyük olduğunda yeniden yapılandırma süresinin ek yükü yüksektir. Önerilen tasarım yalnızca bir bölüme sahip olduğundan, ağ n katmana sahipse, genel gider süresi n× yeniden yapılandırma süresidir. ZyCAP yüksek verimli bir yeniden yapılandırma denetleyicisi olarak benimsenmesine rağmen, uzun yeniden yapılandırma süreleri yine de genel çıktı performansını etkilenmektedir.

Bu sorunu çözmek için ağ bölümleme yöntemi önerilmiştir. Bu yöntem, donanım yapılandırmalarının sayısını azaltmak için ConNN ‘deki katmanları benzer yapısal özelliklere sahip gruplandırma tekniğidir. Katman parametresine ve veri miktarına göre katmanlar arasındaki ilişki tanımlanır. Katmanların benzerliğini belirlemek için iki ilgili faktör kullanılır. İlk faktör, arabellek boyutlarının benzerliğidir. Örneğin, K, N ve K×N'nin değeri yakınsa, bu katmanlar, girdi verileri için aynı boyutta bellek kaynakları gerektirme eğiliminde olan girdi verilerinin benzer bir yapısına sahiptir. Benzer şekilde M, K ve M×K değerleri birbirine yakınsa, ağırlıklar için aynı arabellek boyutlarını gerektirme eğilimindedir. Diğer bir faktör, bilgi işlem kaynakları için gereksinimlerin benzerliğidir. Hesaplama modüllerinin sayısı, N ve K parametrelerinin değerlerine bağlıdır. Dolayısıyla, N ve K'nin değeri yakınsa, aynı boyuttaki hesaplama bloklarıyla yüksek hesaplama verimliliği elde etme olasıdır.

Arabellek boyutlarının benzerliğine dayalı olarak ağdaki katmanların alt kümelerini belirlemek için K-ortalama kümeleme algoritması (Jain, 2010) benimsenmiştir. Kümeleme için kullanılan özellikler, katman parametreleri (M, N, K) ve katmandaki çıktı (M×N), girdi (N×K) ve ağırlık (M×K) verilerinin boyutlarıdır. Euclidean (Öklidyen) uzaklık hesaplaması, özelliklerin benzerliğini ölçmek için kullanılır. Katmanları kümeler halinde gruplandırdıktan sonra, döngü döşeme değişkenlerinin (T_m, T_n, T_k) ve döngü açma değişkenlerinin (U_m, U_n, U_k) kümedeki katmanlar için en uygun şekilde ayarlandığı bir donanım konfigürasyonu tasarlanabilir.

4.4.4. Bölüm Eşleme

Statik yapılandırma hızlandırıcısı, ağdaki katmanlar farklı şekilde yapılandırıldığından yetersiz kullanım sorunuyla karşı karşıyadır. Bu sorunu analitik olarak ele almak için bir arabellek yetersiz kullanımı parametresi (Buffer underutilization: BU) önerilmiştir. BU değişkeni, belirli bir bellek kaynağının ne kadar yetersiz kullanıldığını değerlendirmek için bir ölçümdür. BU göstergesi Denklem (4.1)'deki gibi elde edilebilir.

$$BU = \left(\left\lceil \frac{M}{T_m} \right\rceil - \frac{M}{T_m}\right) + \left(\left\lceil \frac{N}{T_n} \right\rceil - \frac{N}{T_n}\right) + \left(\left\lceil \frac{K}{T_k} \right\rceil - \frac{K}{T_k}\right) \quad (4.1)$$

BU puanı, katman parametreleri ve döşeme değişkenleri arasında bölünerek hesaplanan artık bir değerdir. Başka bir deyişle, hızlandırıcı her katman için verileri yükledikten sonra kalan arabellek öğesinin hesaplanmasına imkan verir. Hızlandırıcıdaki arabellekler tamamen kullanıldığında BU puanı sıfırdır. BU sıfırdan büyükse arabelleklerin bazı bölümlerinin kullanılabilir olduğunu ancak kullanımda olmadığı anlaşılır. Bu nedenle, hızlandırıcıda oluşturulan arabellekler çok iyi kullanılmamaktadır. Amacımız, bellek kaynağı kullanımının maksimum verimliliğini elde etmek için BU puanını en aza indirmektir. Minimize edilmiş BU parametresi, katman sayısına eşit küme sayısı verilerek ve ardından her katman için döşeme ve açma değişkenlerini ayrı ayrı ayarlayarak elde edilebilir. Ancak tam kümelemenin dezavantajı, genel yürütme süresine hâkim olabilen yüksek yeniden yapılandırma süresidir. Kümedeki katmanları kolaylaştırmak adına yapılandırmayı değiştirmek için kullanılan yeniden yapılandırma süresinin maliyetinden kaçınılamaz. Bu nedenle, optimal küme sayısını seçmek için kaynak verimliliği ile

birlikte yeniden yapılandırma süresi dikkate alınmıştır. Toplam yeniden yapılandırma süresi (T_{pr}) Eşitlik (4.2)'deki gibi yazılabilir.

$$T_{pr} = \frac{PB_{boyut} \times N_{pr}}{R_{hız}} \tag{4.2}$$

Burada PB_{boyut}, kısmi bit akışı dosyasının (MByte) boyutudur. N_{pr}, ConNN modelinde kümeleme için gereken yeniden yapılandırma sayısıdır. $R_{hız}$, denetleyicinin yeniden yapılandırma çıktısıdır (MByte/s).

Gereken yeniden yapılandırma miktarı, modeldeki katmanların nasıl kümelendiğine bağlıdır. Minimum N_{pr}, K - 1'dir. Burada K, küme sayısıdır. Maksimum N_{pr}, n'dir. Burada n, katman sayısıdır. Kısmi bit akışı dosyasının boyutu, yeniden yapılandırılabilir bölümün boyutuna bağlıdır. Büyük bir RP, paralel işleme ve veri arabelleğe alma için yeterli kaynak sağlar. Ancak kısmi bit akışı dosyasının boyutu büyük olabilir. Statik hızlandırıcı tasarımında, mevcut kaynakların sayısı FPGA'nın model spesifikasyonu ile sınırlıdır. Tasarımcılar donanım modüllerinin mevcut kaynaklar içinde nasıl optimize edileceğine odaklanabilir. Kısmi yeniden yapılandırma ise, kullanıcının kat planlama sürecinde bölme boyutunu manuel olarak atadığı RP'den oluşur.

RP'nin boyutunu etkili bir şekilde belirlemek için iki faktörü dikkate alınmıştır: hızlandırıcı için mevcut kaynakların sayısı ve yeniden yapılandırma süresi ek yükü. Bu çalışmada, hızlandırıcı için kaynakları kolaylaştırmak adına tek bir RP kullanılmıştır. Hızlandırıcının tüm konfigürasyonları aynı kaynak kısıtlamaları altında tasarlanmıştır. Bu nedenle, kaynak miktarı, en büyük RM'yi barındıracak kadar büyük olmalıdır. Bu çalışmada, kısmi bit akışı dosyasının boyutu, tam bit akışı dosyasının yarısından daha az olan 1.5 MB olarak atanmıştır. Ancak bu boyutta, RP, hızlandırıcının umut verici bir verimle gerçekleştirmesi için yeterli kaynakları sağlayabilir.

Bu nedenle, optimal küme sayısını bulmak adına tüm olası kümeler için BU ve T_{pr} değerleri hesaplanır ve sonra minimum BU ve T_{pr}'yi sağlayan küme sayısını seçilir. Bu şekilde, donanım yapılandırmalarının miktarı sınırlandırılır ve yeniden yapılandırma süresinin genel çıktı performansı üzerinde daha az etkisi olur.

4.5. Deneysel Sonuçlar

Önerilen tasarımın etkinliğini göstermek için AlexNet, ResNet-18 ve YOLOv3-tiny dahil olmak üzere çeşitli ConNN modelleri uygulanmıştır. İlk olarak, katmanları kümeye eşlemek için ağ bölümleme yöntemi kullanılmıştır. Sonraki adım, her küme için en uygun donanım yapılandırmasını bulmaktır. Optimal küme sayısını (K) elde etmek için BU'yu ve yeniden yapılandırma süresini hesaplayan ağ eşleme stratejisi sunulmuştur. Optimal küme sayısını elde ettikten sonra optimum donanım yapılandırması ile her küme için yeniden yapılandırma modülü oluşturulmuştur. Daha sonra hızlandırıcının kaynak kullanımı ve hızlandırıcı verimliliği sunulmuştur.

4.5.1. Ağ Bölümleme Sonuçları

Ağ bölme yöntemi, Python programlama dilinde, Sklearn yazılım kütüphanesinden K-ortalama kümeleme yöntemiyle uygulanmıştır. Tüm olası küme sayılarını kullanırken AlexNet, ResNet-18 ve YOLOv3-tiny'nin ağ bölümleme sonuçları Tablo 4.1, Tablo 4.2 ve Tablo 4.3 'te sırasıyla özetlenmiştir. Sütun K kümelerin sayısını belirtir ve alt simge ile C değişkeni küme etiketini belirtir.

Bu sonuçlar, benzer yapısal özelliklere sahip katmanların bir arada kümelendiğini göstermiştir. Modelin girdisine yakın katmanlar aynı grupta sınıflandırılır çünkü bu katmanlar girdi verilerinin boyutunun ağırlıklardan çok daha büyük olduğu benzer bir yapıya sahiptir. Aynı şekilde daha derindeki katmanlar bu katmanlardaki verilere ağırlıklar hâkim olduğundan aynı grupta kümelenmiştir. Ayrıca ilk katmanın yapısının diğer katmanlardan önemli ölçüde farklı olduğu da fark edilmiştir çünkü bu katmandaki girdi verileri orijinal görüntünün pikselleridir.

Tablo 4.1. AlexNet ‘in ağ bölümleme sonuçları

Katman	Küme sayısı (K)			
	2	3	4	5
Conv1	C_1	C_1	C_1	C_1
Conv2	C_2	C_2	C_2	C_2
Conv3	C_2	C_3	C_3	C_3
Conv4	C_2	C_3	C_3	C_4
Conv5	C_2	C_3	C_4	C_5

Tablo 4.2. ResNet-18'in ağ bölümleme sonuçları

Katman	Küme sayısı (K)							
	2	3	4	5	6	7	8	9
Conv1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1
Conv2	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2
Conv3	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2
Conv4	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2
Conv5	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2
Conv6	C_1	C_2	C_3	C_3	C_3	C_3	C_3	C_3
Conv7	C_1	C_2	C_3	C_3	C_3	C_3	C_4	C_4
Conv8	C_1	C_2	C_3	C_3	C_3	C_3	C_4	C_4
Conv9	C_1	C_2	C_3	C_3	C_3	C_3	C_4	C_4
Conv10	C_2	C_2	C_3	C_3	C_4	C_4	C_3	C_5
Conv11	C_2	C_2	C_3	C_3	C_4	C_4	C_5	C_6
Conv12	C_2	C_2	C_3	C_3	C_4	C_4	C_5	C_6
Conv13	C_2	C_2	C_3	C_3	C_4	C_4	C_5	C_6
Conv14	C_2	C_3	C_3	C_4	C_4	C_5	C_6	C_7
Conv15	C_2	C_3	C_4	C_5	C_5	C_6	C_7	C_8
Conv16	C_2	C_3	C_4	C_5	C_5	C_6	C_7	C_8
Conv17	C_2	C_3	C_4	C_5	C_5	C_6	C_7	C_8
Conv18	C_2	C_3	C_3	C_4	C_6	C_7	C_8	C_9

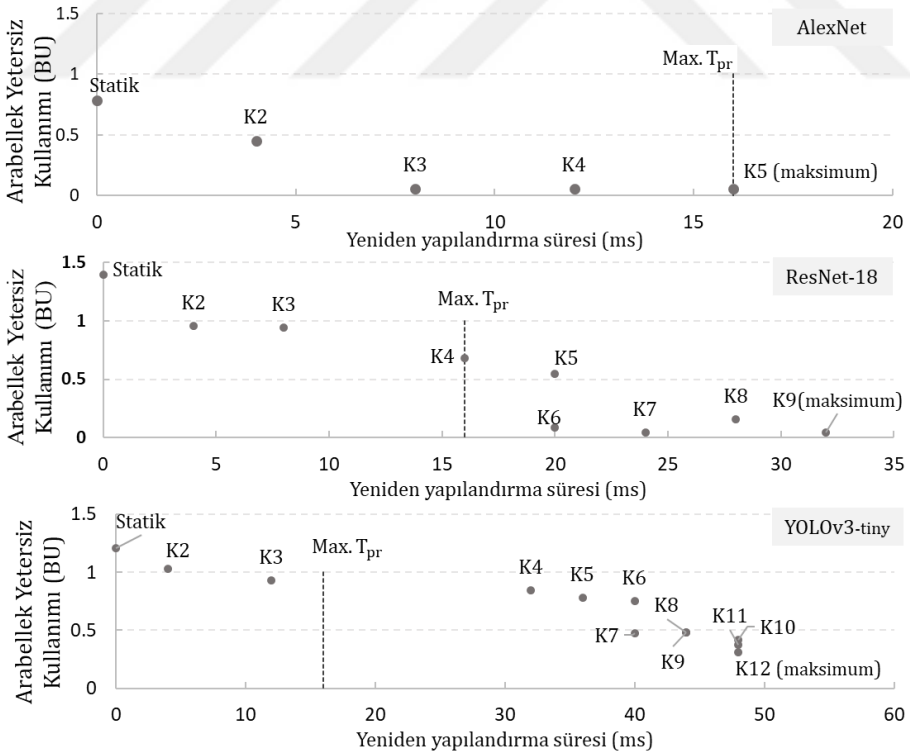
Tablo 4.3. YOLOv3-tiny'nin ağ bölümleme sonuçları

Katman	Küme sayısı (K)										
	2	3	4	5	6	7	8	9	10	11	12
Conv1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1	C_1
Conv2	C_1	C_1	C_1	C_2	C_2	C_2	C_2	C_2	C_2	C_2	C_2
Conv3	C_2	C_2	C_2	C_2	C_3	C_3	C_3	C_3	C_3	C_3	C_3
Conv4	C_2	C_2	C_2	C_3	C_4	C_3	C_4	C_4	C_4	C_4	C_4
Conv5	C_2	C_2	C_2	C_3	C_4	C_4	C_5	C_5	C_5	C_5	C_5
Conv6	C_2	C_2	C_3	C_4	C_5	C_5	C_6	C_6	C_6	C_6	C_6
Conv7	C_2	C_3	C_4	C_5	C_6	C_6	C_7	C_7	C_7	C_7	C_7
Conv8	C_2	C_2	C_2	C_3	C_4	C_4	C_5	C_5	C_5	C_8	C_8
Conv9	C_2	C_2	C_3	C_4	C_5	C_5	C_6	C_6	C_6	C_9	C_6
Conv10	C_2	C_2	C_2	C_3	C_4	C_4	C_5	C_8	C_8	C_{10}	C_9
Conv11	C_2	C_2	C_2	C_3	C_4	C_4	C_5	C_8	C_9	C_{11}	C_{10}
Conv12	C_2	C_2	C_3	C_4	C_5	C_7	C_8	C_9	C_{10}	C_{12}	C_{11}
Conv13	C_2	C_2	C_2	C_3	C_4	C_4	C_5	C_8	C_8	C_{10}	C_{12}

Maksimum K sayısı modelden modele değişir. AlexNet modeli 5 farklı evrişimsel katmanına sahiptir. Bu model 5 katman grubuna ayrılabilmiştir. ResNet-18 için bu modelde katmanların tekrar tekrar kullanılması nedeniyle maksimum küme sayısı 9'dur. YOLOv3-tiny, 2 katmanın yapısal olarak aynı olduğu 13 evrişim katmanından oluşur. Bu model için maksimum küme sayısı 12 kümedir. Kümeleme yönteminin sonuçlarını elde ettikten sonra en düşük hesaplama gecikmesini ve minimum bellek erişimini elde etmek için en uygun donanım yapılandırması hesaplanmıştır.

4.5.2. Bölüm Eşleme Sonuçları

AlexNet, ResNet-18 ve YOLOv3-tiny için donanım yapılandırma değerlendirmelerinden elde edilen tüm olası BU ve T_{pr} değerleri Şekil 4.6'da listelenmiştir. X-ekseni, ConNN çıkarımı sırasında meydana gelen toplam yeniden yapılandırma süresini (T_{pr}) temsil eder ve Y-ekseni BU değerini temsil eder. Kesikli çizgi, yeniden yapılandırma giderlerinin genel performansı önemli ölçüde etkilememesini sağlamak için tanımlanan yeniden yapılandırma süresinin maksimum değerini gösterir.



Şekil 4.6. Farklı sayıda küme kullanıldığında yetersiz arabellek kullanımı parametresi (BU) ve yeniden yapılandırma süresi

Sonuçlar, statik tasarımı kullanmanın en yüksek BU puanını verdiğini göstermiştir. Ağ kümeleme yöntemiyle birden çok donanım yapılandırması kullanıldığında BU puanı önemli ölçüde azalır. Ayrıca küme sayısının arttırılması BU değerini daha da düşürür. Ancak BU ile yeniden yapılandırma süresi arasında bir ödünleşim vardır. Birçok küme BU puanını büyük ölçüde düşürür ancak çok yüksek yeniden yapılandırma süreleri nedeniyle üretim performansı düşer. Çıktı performansı ve kaynak verimliliği arasında bir denge sağlamak için hızlandırıcının gerçek zamanlı video kare hızında çalışmasına izin veren yürütme süresinin yarısı olarak hesaplanan maksimum yeniden yapılandırma süresi olan 16 ms atanmıştır. Bu nedenle, 16 ms'den daha büyük bir yeniden yapılandırma süresi gerektiren K kümeleri dikkate alınmayacaktır.

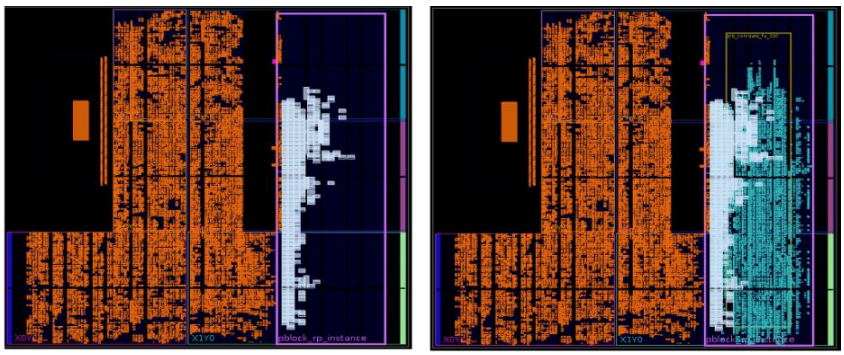
Bu sonuç, en düşük BU'yu veren K sayısını seçmek için incelenmiştir. Birden fazla nokta minimum BU'ya ulaşırsa, yeniden yapılandırma süresi en düşük olan nokta seçilir. Örneğin, AlexNet'in sonuçları, 3 küme (K3), 4 küme (K4) ve 5 kümenin (K5) kullanılmasının zaman kısıtlamaları altında aynı minimum BU'yu sağladığını göstermiştir. Bu şekilde, AlexNet modeli için en uygun küme sayısı olarak 3 küme (K3) seçilmiştir çünkü K4 ve K5'e kıyasla en düşük yeniden yapılandırma süresini almıştır. Resnet-18 sonucunda 6, 7 ve 9 küme kullanıldığında minimum BU elde edilmiştir. Ancak bu noktada toplam yeniden yapılandırma süresi maksimum yeniden yapılandırma süresini aşmıştır. Bu nedenle K4 sınırlı bir yeniden yapılandırma süresi altında en düşük BU'yu sağladığından ResNet-18 modelini 4 kümeye eşlenmiştir. YOLOv3-tiny için 3 kümenin (K3) kullanılması yeniden yapılandırma süresi kısıtlamaları altında minimum BU'ya ulaşmıştır. Bu nedenle, YOLOv3-tiny katmanları 3 kümeye eşlenmiştir.

Statik yapılandırma, yeniden yapılandırma süresi gerektirmese de kaynakları çok verimli kullanmaz. Ancak kümeleme yöntemiyle dinamik yapılandırmalar, statik yapılandırmaya kıyasla kabul edilebilir yeniden yapılandırma zaman yükü ile kaynak verimliliğini artırmıştır. Optimal küme sayısına ulaştıktan sonra, donanım hızlandırıcı kümedeki katman özelliklerine göre tasarlanmıştır.

4.5.3. Kaynak Kullanımı

Kullanılabilir kaynakların sayısı, FPGA kat planındaki yeniden yapılandırılabilir bölümün şekline bağlıdır. 1,5 MB'lık bit akışı dosyası altında yeterli BRAM blokları ile

maksimum sayıda DSP ve LUT elde etmek için yeniden yapılandırılabilir bölüm FPGA kat planında Şekil 4.7 'deki gibi çizilir.



Statik bölüm içinde sabit tasarım RP içinde yeni tasarım

Şekil 4.7. FPGA' da statik ve yeniden yapılandırılabilir bölümler

RP, kat planının en üstünden sağ altına doğru FPGA'ya eşlenmiştir. Seçilen bölüm, yeterli sayıda DSP ve BRAM kaynağı sağlamıştır. Sonuç olarak yeniden yapılandırılabilir bölüm, 112 DSP, 11440 LUT ve 18 KB BRAM'lik 112 blok dahil olmak üzere FPGA'nın kaynaklarını sağlamıştır. Hızlandırıcının donanım modülü, bu kaynak kısıtlamaları altında uygulanmıştır. AlexNet, ResNet-18 ve YOLOv3-tiny için statik ve uyarlanabilir hızlandırıcıların yapılandırma değişkenleri ve kaynak kullanım sonuçları Tablo 4.4, Tablo 4.5 ve Tablo 4.6'da sırasıyla listelenmiştir.

Tablo 4.4. AlexNet 'in kaynak kullanım sonuçları

Küme	T_m	T_n	T_k	U_n	#BRAM	#DSPs	#FFs	#LUTs
Statik	64	243	9	11	90	109	12202	11988
C1	64	440	11	9	54	101	12182	10649
C2	48	440	11	9	90	109	12493	10957
C3	128	184	23	4	56	102	12052	9561

Tablo 4.5. ResNet-18'in kaynak kullanım sonuçları

Küme	T_m	T_n	T_k	U_n	#BRAM	#DSPs	#FFs	#LUTs
Statik	128	352	11	9	90	109	12636	10403
C1	64	352	11	9	90	109	12636	10403
C2	32	684	12	8	90	106	12184	11063
C3	256	64	8	12	82	106	12113	11026
C4	32	512	32	3	80	106	11464	8809

Tablo 4.6. YOLOv3-tiny'nin kaynak kullanım sonuçları

Küme	T_m	T_n	T_k	U_n	#BRAM	#DSPs	#FFs	#LUTs
Statik	128	176	11	9	90	109	12428	10500
C1	128	176	11	9	90	109	12428	10500
C2	256	88	11	9	90	109	12562	10527
C3	16	1056	33	3	64	109	11500	9092

Sonuçlar, her kümenin farklı sayıda kaynak gerektirdiğini göstermektedir. DSP'lerin kullanımı, beklenen işlem hızına ulaşmak için bazı kümelerde daha fazla bilgi işlem kaynağına ihtiyaç olduğunu göstermiştir. Katmanlar farklı sayıda veri içerir. Bu da bazı kümeler için daha düşük BRAM kullanımını sağlamıştır. Uyarlanabilir hızlandırıcının kaynak kullanımı Tablo 4.7'de özetlenmiştir.

Tablo 4.7. Uyarlanabilir hızlandırıcının kaynak kullanımı

Model	#BRAM	#DSPs	#FFs	#LUTs
AlexNet	63	104	12166	10058
ResNet-18	88	108	12383	10565
YOLOv3-tiny	86	109	12295	10285

4.5.4. İşlem Oranı Performansı

Tablo 4.8'de uyarlanabilir hızlandırıcının işlem oranı performansı ve statik hızlandırıcı ile performans karşılaştırması rapor edilmektedir. Sonuçlarda, en yüksek işlem oranı (peak GOP/s) ve ortalama işlem oranı (Avg. GOP/s) gösterilmiştir. Uyarlanabilir hızlandırıcı için yeniden yapılandırma süresi de dahil olmak üzere ortalama işlem oranı sunulmuştur.

Tablo 4.8. FPGA üzerinde hızlandırıcının işlem oranı performansı

Model	Statik tasarım (GOP/s)		Uyarlanabilir tasarım (GOP/s)		
	Peak	Avg.	Peak	Avg.	T_{pr} ile Avg.
AlexNet	34,6	30,25	38,9	34,1	30,2
ResNet-18	32,4	28	35,3	31,1	28,1
YOLOv3-tiny	34,8	28,2	36,4	31,1	28,8

Sonuçlar, uyarlanabilir hızlandırıcının sırasıyla AlexNet, ResNet-18 ve YOLOv3-tiny üzerinde değerlendirildiğinde statik tasarıma kıyasla 1,09×, 1,06× ve 1,02× daha iyi peak GOP/s elde ettiğini göstermiştir. Uyarlanabilir hızlandırıcı, statik şemaya kıyasla daha yüksek ortalama işlem oranı (Avg. GOP/s) da elde etmiştir. Bu sonuçlar dinamik

donanımın ConNN hızlandırıcısı için en uygun donanım yapılandırmasını sağlayabildiğini doğrulamaktadır. Sonuçlar ayrıca uyarlanabilir hızlandırıcının işlem oranı performansının, genel yeniden yapılandırma süresi nedeniyle azaldığını göstermektedir. Ancak yeniden yapılandırma süresiyle, uyarlanabilir hızlandırıcı, statik hızlandırıcı ile aynı düzeyde ortalama işlem oranı performansı elde etmiştir.

4.6. Performans Karşılaştırması

Uyarlanabilir ve statik hızlandırıcıların performans karşılaştırmaları Tablo 4.9'da özetlenmiştir. Sonuçlar, uyarlanabilir hızlandırıcının statik tasarıma benzer kaynak verimliliği sağlayabileceğini göstermiştir. BU değerlendirme sonuçlarında, uyarlamalı hızlandırıcı, AlexNet, ResNet-18 ve YOLOv3-tiny için statik tasarım üzerinde kaynağın yetersiz kullanımında sırasıyla 13,4×, 2,06× ve 1,3× iyileştirmeler sunmuştur. Ayarlanabilir arabellek boyutunun kullanılmasıyla, uyarlanabilir tasarım, statik şemaya kıyasla BRAM bloklarının sayısını azalmıştır. Bu sonuçlar, hızlandırıcının yüksek verimli kaynak kullanımını sağlamak için dinamik donanım tasarımının kullanılabileceğini kanıtlamıştır.

Tablo 4.9. Statik hızlandırıcı ile performans karşılaştırması

Parametre	AlexNet		ResNet-18		YOLOv3-tiny	
	Statik	Uyabilen	Statik	Uyabilen	Statik	Uyabilen
#LUTs	11988	10058	10403	10565	10500	10285
#DSPs	109	103	109	108	109	109
#BRAM	90	63	90	88	90	86
#FFs	12202	12166	12636	12383	12428	12295
GOP/s	30,25	30,28	28	28,14	28,2	28,8
GOP/s/DSP	0,28	0,29	0,26	0,26	0,26	0,26
GOP/s/kLUT	2,52	3,01	2,69	2,66	2,69	2,8
BU	0,78	0,058	1,4	0,68	1,21	0,93
T_{pr}	-	8 ms	-	16 ms	-	12 ms

Genel olarak, uyarlanabilir hızlandırıcı, statik hızlandırıcıya benzer kaynak kullanımını sağlamıştır, ancak uyarlanabilir tasarım, statik tasarımdan daha düşük bellek kaynağı kullanımını gerektirmiştir. Yeniden yapılandırma süresinin, işlem oranı ve kaynak verimliliği üzerinde önemli bir etkisi yoktur. Daha da önemlisi, önerilen tasarım hızlandırıcı mimarisi için uyarlanabilirlik sunmuştur. Bu nedenle, hızlandırıcının yapısı

optimum donanım konfigürasyonu ile güncellenebilir. Önerilen yöntem, statik tasarım kullanılarak çözülemeyen yetersiz kullanım sorununu iyileştirmeyi amaçlamaktadır. Bu özellik, statik yapılandırma tasarımı ile çözülemeyen yetersiz kullanım sorununu iyileştirmeyi amaçlamıştır.

4.7. Sonuçlar

ConNN mimarisi, hesaplama iş yüklerinin sayısı ve veri boyutunun farklı olduğu çeşitli özelliklere sahip yığın katmanlardan oluşur. Statik hızlandırıcı, bazı kaynaklar mevcut olduğunda ancak kullanımda olmadığında kaynakların yetersiz kullanımıyla karşı karşıya kalır. Bunu geliştirmek için kısmi yeniden yapılandırma kullanılarak uyarlanabilir bir hızlandırıcı önerilmiştir. ConNN modelini katmanın yapısal benzerliğine dayalı olarak bir kümeye eşlemek için bir ağ bölme yöntemi açıklanmıştır. Sonuçlar, uyarlanabilir hızlandırıcının statik hızlandırıcıya kıyasla kaynakların yetersiz kullanımını büyük ölçüde iyileştirdiğini göstermiştir.

5. YÜKSEK ÇÖZÜNÜRLÜKLÜ LOGARİTMİK NİCELEME

Kaynak açısından verimli donanım hızlandırıcısı önceki bölümde verilmiştir. Bu bölümde, model sıkıştırma tekniği özellikle bir nicemleme yöntemi, veri kesinliğini kayan noktadan 32 bitin altındaki bitlere indiren bir teknik olarak gösterilmiştir. Elde edilen doğruluk sonucunda meydana gelen küçük bir azalma ile çok düşük bit genişlikli veri hassasiyeti kullanarak ConNN çıkarımını gerçekleştirebilen verimli, donanım dostu bir nicemleme yöntemi sunulmuştur. FPGA üzerinde ConNN çıkarımını hesaplamak için çarpansız bir hızlandırıcı önerilmiştir. Hızlandırıcının performansını ve nicemleme verimliliğini içeren deneysel sonuçlar rapor edilmiştir.

5.1. Tanıtım

Kayan nokta temsili, ConNN modelinde ağırlık değerini ve özellik haritaları temsil etmek için kullanılan bir veri türüdür. Kayan nokta temsili, bir tamsayı ve toplam bit sayısının 32 bit olduğu bir kesir bölümünden oluşur (tek kesinlik olarak da bilinir). IEEE754 formatında, bir tamsayı ve bir kesir kısmı sırasıyla 8 bit ve 23 bit ile sunulur. Sayısal cihazlardaki bellek blokları ve aritmetik operatörler, 32 bit veri hassasiyetini desteklemek üzere tasarlanır. FPGA'lar gibi gömülü cihazlarda 32-bit ConNN modeli uygulamak zor bir iştir. Çok sayıda bilgi işlem kaynağı ve bellek gerektirir. Bu sorunu çözmek için bir veri nicemleme yöntemi önermiştir.

ConNN çalışmasında, nicemleme, 32-bit tabanlı bir veriyi orijinal değer üzerinde çok az etkiyle daha düşük bitlere dönüştürmek için kullanılan bir tekniktir. Nicemleme tarafından sunulan değerlere nicemleme seviyeleri denir. İki nicemleme seviyesi arasındaki boşluk, adım boyutu olarak bilinir. Nicemleme yönteminin kalitesi, nicemleme düzeylerinin sayısına ve adım boyutuna bağlıdır. Küçük bir adım boyutuna sahip çok sayıda nicemleme seviyesi, yüksek kaliteli bir nicemleme sağlar. Nicemleme düzeylerinin sayısı, bit genişliğinin sayısıyla tanımlanır. Bu nedenle, büyük bir bit genişliği daha iyi nicemleme kalitesi sağlar. Düzgün nicemleme ve Düzgün olmayan nicemleme olma üzere iki tür nicemleme yöntemi vardır. Düzgün nicemleme, seviyeler boyunca eşit adım boyutuna sahip nicemleme seviyeleri sunar. Düzgün olmayan yöntem ise eşit olmayan adım boyutuna sahiptir.

ConNN modelinin daha düşük bir bit genişliğine nicemleme, veri hassasiyetinin düşük olması nedeniyle doğrulukta bozulmaya neden olabilir. Tatmin edici bir doğruluk elde etmek için nicelleştirilmiş bir ConNN modelini sıfırdan eğitmek veya önceden eğitilmiş bir modelden yeniden eğitmek gerekir. Ancak bu yeniden eğitim süreci zaman alıcı olabilir ve ek hesaplamalar gerektirir. Ayrıca bazı orijinal eğitim veri kümeleri, gizlilik veya mülkiyet nedeniyle kamuya açık değildir. Örneğin, özel şirketler veya sağlık sistemi veri kümeleri tarafından toplanan müşteri verileri vb gibi.

32-bit ConNN modelini yeniden eğitim süreci olmadan daha düşük bitlere sıkıştırmak için bir eğitim sonrası nicemleme yöntemi önerilmiştir (Banner ve diğ., 2019; Y. Cai ve diğ., 2020; R. Zhao ve diğ., 2019). Eğitim sonrası nicemlemedeki çalışmaların çoğu doğrusal bir nicemleme yöntemidir. Doğrusal nicemleme, nicemleme adımının eşit aralıklarla yerleştirildiği tek tip nicemleme yöntemlerinden biridir. Doğrusal şema için bit genişliği sayısı artırılarak veya azaltılarak adım boyutu ayarlanabilmiştir. Doğrusal nicemleme hesaplamayı basitleştirebilse de eğitim sonrası şemada bit genişliği sayısı 8 bitten büyük olduğunda daha iyi çalışmıştır. ConNN modellerini sıkıştırmak için logaritmik nicemleme olarak bilinen başka bir verimli nicemleme yaklaşımı önerilmiştir (Miyashita ve diğ., 2016). Birçok çalışma, ConNN modellerindeki ağırlıklar küçük değerler (sıfıra yakın) olduğundan, çok düşük bit genişliğine uygulandığında logaritmik nicemlemenin doğrusal yöntemden daha iyi performans sergilediğini göstermiştir. Ayrıca logaritmik yöntem donanım dostu bir hesaplamadır çünkü logaritmik nicemlenmiş değerlerin çarpımı gömülü aygıtlar için temel bir işlem olan bit kaydırma kullanılarak gerçekleştirilebilir.

Bununla birlikte, ConNN modeline yeniden eğitim olmadan logaritmik bir yöntem uygulamak zorluklar sunar. Ağırlık parametreleri sıfır civarında değilse, logaritmik yöntem yüksek doğrulukta bozulmaya maruz kalabilir. Ayrıca, bit genişliğini artırmak, bu sorunu çözmek için daha iyi çözünürlük sağlamaz. Bu sorunu ele almak için bu tezde nicemleme seviyesinin bir tamsayı yerine gerçek bir sayı ile verildiği, yüksek çözünürlüklü bir logaritmik nicemleme önerilmektedir. Ayrıca basit bir bit kaydırma işlemi kullanarak nicemlenmiş ConNN modellerini gerçekleştirmek için verimli bir hızlandırıcı ortaya geliştirilmiştir.

5.2. İlgili Çalışmalar

Bu kısımda literatürdeki ilgili çalışmaların bir incelemesi sunulmaktadır. Logaritmik ölçeklerle nicemleme, ConNN modelini çok düşük bit genişliğiyle sıkıştırabilen etkili yöntemlerden biri olarak gösterilmiştir. Özellik haritaları ve ağırlıkları 4 bite nicemleme için (Miyashita ve diğ., 2016)'da logaritmik nicemleme önerilmiştir. Ortaya çıkan sonuçlar, logaritmik kuantizasyonun doğrusal nicemleme yönteminden daha iyi performans gösterdiğini ve doğruluk bozulmasının sadece yüzde birkaç olduğunu göstermiştir. Ancak yeniden eğitim olmadan logaritmik şema kullanmak, bir dengesizlik çözme sorunu nedeniyle doğruluğu önemli ölçüde azaltabilir.

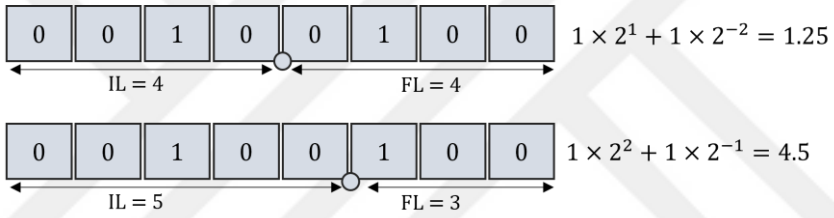
Bu sorunu ele almak için (S. E. Chang ve diğ., 2021; Y. Li ve diğ., 2019), yüksek girdilerde çözünürlüğü iyileştirmek için ölçeklenebilir parametrelerle ikinin kuvveti değerlerinin eklenmesi yoluyla logaritmik tabanlı bir nicemleme önermiştir. Birkaç çalışma, logaritmik temelleri keşfederek kuantizasyonun çözünürlüğünü geliştirmiştir. (Xu ve diğ., 2020)'de bir logaritmik taban $\sqrt{2}$ önerilmektedir. Yazarlar, girdi yüksek olduğunda yüksek çözünürlük tabanın $\sqrt{2}$ taban-2'den daha iyi performans gösterdiğini söylemektedir. Log-tabanı üzerine daha fazla çalışma (Vogel ve diğ., 2018)'de sunulmuştur. Yazarlar taban-2, taban-$\sqrt{2}$ ve taban-$\sqrt[4]{2}$ olmak üzere farklı logaritmik ölçekler önermiştir. Bu sonuçlar, ağırlıklar için nicemleme taban-$\sqrt{2}$ ve aktivasyonlar için taban-2'nin bir kombinasyonu kullanılarak doğruluğun iyileştirildiğini göstermiştir.

(J. Cai ve diğ., 2018), nicemlenmiş değerleri tamsayılar yerine gerçek sayılar kullanarak ifade etmenin bir yöntemini ifade etmeyi önermektedir. Bu şekilde, iki bitişik nicemleme seviyesi arasındaki adım boyutu, bir yuvarlama işlevi tarafından gerçekleştirilen tam sayıların kullanımından çok daha küçüktür. Adım boyutunun eşit olduğu tek tip nicemleme yönteminden yararlanmak için yazarlar (Huang ve diğ., 2021), logaritmik nicemleme aktivasyonları için sabit noktalı şeması kullanmıştır. Kayan-nokta modellerini yeniden eğitmeden 8 bitten daha az bit genişliğine sıkıştırmayı başarmışlardır.

5.3. Genel Bilgiler

5.3.1. FPGA'da Sabit Noktalı Veri Gösterimi

Sabit nokta gösterilimi, bir tamsayı biti ve bir kesir biti ile bir sayıyı temsil eden bir veri biçimidir. FPGA, kullanıcıların verileri temsil etmek için hem tamsayı hem de kesir bitlerinin bit genişliği sayısını tanımlayabilecekleri sabit nokta adı verilen özel bir veri türü sağlar. Kayan-nokta yerine sabit nokta biçimini kullanmanın amacı, bellek ve bilgi işlem kaynakları gereksinimini azaltmaktır. FPGA'da sabit nokta formatı ap_fixed (W, IL) ile gösterilir. Burada W toplam bit genişliğini ve IL bir tamsayı bitini belirtir. Bu nedenle, bir kesir biti (FL), W - IL ile hesaplanır. Bunu göstermek için sabit noktalı sayıların örnekleri Şekil 5.1'de gösterilmiştir. Basitleştirmek için işaretsiz bir sayının sabit nokta biçimi verilmiştir. İşaretli bir sayı için ilk bit (en soldaki) işaret biti olarak kullanılır.



Şekil 5.1. 8 bitlik sabit noktalı sayı örneği

5.3.2. Doğrusal Nicemleme Yöntemi

Doğrusal nicemleme, bir nicemleme seviyesinin eşit aralıklı olduğu bir tip nicemleme sınıfıdır. Çoğu ConNN nicemleme, 16-bit'in altında yüksek performans elde ettiği için bu yöntem kullanır. Bir kayan-noktayı sabit noktaya nicemlemek için doğrusal nicemleme işlevi Eşitlik (5.1)'deki gibi ifade edilebilir.

$$Q_{lin.}(x) = \text{clip}(\text{round}\left(\frac{x}{\Delta}\right) \cdot \Delta, -2^{FSR}, 2^{FSR} - \Delta) \tag{5.1}$$

Burada x, 32 bitlik bir giriş kayan nokta değeridir. FSR, Tam ölçek aralığı değerini belirtir ve W, toplam bitleri belirtir. Nicemleme adım boyutu aşağıdaki denklem olarak ifade edilebilir.

$$\Delta = 2^{FSR+1-W} \tag{5.2}$$

Bu çalışmada FSR değerini şu şekilde seçmiştir: $2^{\mathrm{FSR}} = 2^{\mathrm{IL}-1}$. Böylece nicemleme adım boyutu şu şekilde yazılabilir: $\Delta = 2^{-\mathrm{FL}}$.

Bir kırpma işlevi (clip) Denklem (5.3)'teki gibi ifade edilebilir:

$$\mathrm{clip}(\mathrm{x}, \min, \max) = \begin{cases} \min, & \mathrm{x} \leq \min \\ \mathrm{x}, & \min < \mathrm{x} < \max \\ \max, & \mathrm{x} \geq \max \end{cases} \tag{5.3}$$

Nicemlemenin kalitesi, bit genişliğinin sayısına bağlıdır. Bit genişliğini doğrusal olarak artırmak, nicemleme düzeyinin sayısını artırır ve daha yüksek bir çözünürlük verir. Ancak, daha fazla bellek alanı gerektirir ve hesaplama maliyeti pahalıdır. Tersine, daha düşük bit genişliğinin kullanılması daha az bellek gerektirir ancak doğruluk oranı düşebilir.

5.3.3. Logaritmik Nicemleme Yöntemi

Bu yöntem, logaritmik ölçekteki değeri kullanarak giriş değerini çıkış değerine eşitler. Logaritmik nicemleme yöntemi, nicemleme seviyelerinin eşit olmadığı ve nicemlemenin bir değerini temsil etmek için logaritma hesaplamasını içeren düzgün olmayan bir nicemlemedir. Logaritmik nicemleme parametreleri, FSR ve toplam bit genişliğidir (W). Logaritmik nicemleme Eşitlik (5.4)'teki gibi elde edilebilir.

$$\mathrm{Q}_{\log}(\mathrm{x}) = \mathrm{sign}(\mathrm{x}) \cdot 2^{\tilde{\mathrm{x}}} \tag{5.4}$$

Burada

$$\tilde{\mathrm{x}} = \mathrm{clip}(\mathrm{round}(\log_2|\mathrm{x}|), 2^{\mathrm{FSR}} - 2^{\mathrm{W}}, 2^{\mathrm{FSR}} \tag{5.5}$$

Doğrusal nicemleme yönteminden farklı olarak, logaritmik şema, ortalama etrafında yüksek sayıda nicemleme seviyesi (yüksek çözünürlük) ve diğer bölgeler için daha düşük bir çözünürlük verir. Bu nedenle, ağırlıkların yapısı, ağırlıkların çoğunun sıfır civarında olduğu çan şeklinde bir dağılım olarak kabul edilebileceğinden, ConNN'deki ağırlıklar için logaritmik nicemlemeyi kullanmak iyi bir seçimdir. Logaritmik nicemlemenin avantajı, nicemlenmiş değerler arasındaki bir çarpma işlemi bit kaydırma işlemi ile değiştirilebildiğinden donanım dostu olmasıdır. Örneğin, verilen nicemlenmiş ağırlıklar ($\tilde{\mathrm{w}}$) ve aktivasyonlar (a). Çarpma Denklem (5.6)'da gibi çarpansız yapılabilir.

$$a \times 2^{\tilde{w}} = \begin{cases} a \ll \tilde{w}, & \tilde{w} > 0 \\ a \gg \tilde{w}, & \tilde{w} < 0 \\ a, & \tilde{w} = 0 \end{cases} \quad (5.6)$$

Burada $\ll$ sol kaydırıcı ve $\gg$ sağ kaydırıcıdır.

5.4. Yüksek Çözünürlüklü Logaritmik Nicemleme

Logaritmik nicemleme, ConNN'nin çok düşük bit genişliğiyle nicemleme potansiyelini gösterir. 8 bitten daha düşük bit genişliği ile uygulandığında doğrusal nicemlemeden daha iyi performans gösterir çünkü logaritmik yöntem ortalama etrafında yüksek çözünürlüklü özelliklere sahipken doğrusal şema eşit çözünürlüğe sahiptir. Bununla birlikte, yüksek değerli bir girişte düşük çözünürlük nedeniyle büyük bir mutlak girdisi nicelleştirirken logaritmik yöntem iyi performans göstermez. Örneğin, 2^{-3} ve 2^{-4} seviyeleri arasındaki adım boyutu 0,0625 iken adım boyutu, 2^{0} ve 2^{-1} nicemleme seviyesi arasında 0,5'e yükselir.

Log alanındaki nicemlenmiş değeri temsil etmek için en yakın tam sayıya yuvarlamanın logaritmik yöntemin bir dezavantajı olduğu gözlemlenmiştir. Bu durum, iki nicemleme seviyesi arasında büyük bir adım boyutuna sebebiyet verir. Eşit olmayan adım boyutu sorunu, üstel özellik nedeniyle çözülemez. Ancak, nicemlenmiş değeri temsil etmek için tamsayılar yerine kayan-nokta değerler kullanarak çözünürlüğü artırmak için adım boyutu azaltılabilir. Bu nedenle, ConNN modelinde ağırlıklar ve özellik haritaları için gerçek sayı tabanlı temsili kullanarak logaritmik nicemleme önerilmiştir. Önerilen yöntemin açıklamaları aşağıdaki bölümlerde sunulmuştur.

5.4.1. Qset ile Ağırlıklar İçin Nicemleme Yöntemi

Önerilen yöntem, bir tamsayı değerine yuvarlamak yerine Log alanındaki nicemlenmiş değeri temsil etmek için gerçek sayıları kullanan (J. Cai ve diğ., 2018) çalışmasından esinlenmiştir. Log alanındaki nicemleme seviyeleri için hem tam sayıları hem de gerçek sayıları kullanılarak bu yöntemi daha da geliştirilmiştir. Gerçek sayılar kullanılarak önerilen logaritmik nicemleme yöntemi aşağıdaki gibi yazılabilir.

$$Q_{slog}(w) = \text{sign}(w) \cdot 2^{\tilde{w}} \quad (5.7)$$

Burada

$$\tilde{w} = \text{Qset}[\text{index}] \tag{5.8}$$

Nicemlenmiş ağırlıklar ($\tilde{w}$), log2(x)'e en yakın Qset değeridir. Qset, Kayan-nokta değerini en yakın gerçek değere dönüştüren bir yuvarlama işlevi (rounding function) olarak çalışır. Qset, P1 ve P2 olmak üzere iki alt kümeye sahiptir. P1 kümesi, belirli bir aralıktaki eşit aralıklı gerçek sayılardan oluşur. P2 kümesi, P1'in son elemanından itibaren sayılan tam sayı dizilerinden oluşur. Qset parametreleri, bit genişliği (n), nicemleme aralığı (R) ve set P1'in (S) maksimum temsil düzeyidir. Qset aşağıdaki gibi tanımlanabilir.

$$\text{Qset} = \{\{P_1\}, \{P_2\}\} \tag{5.9}$$

Burada

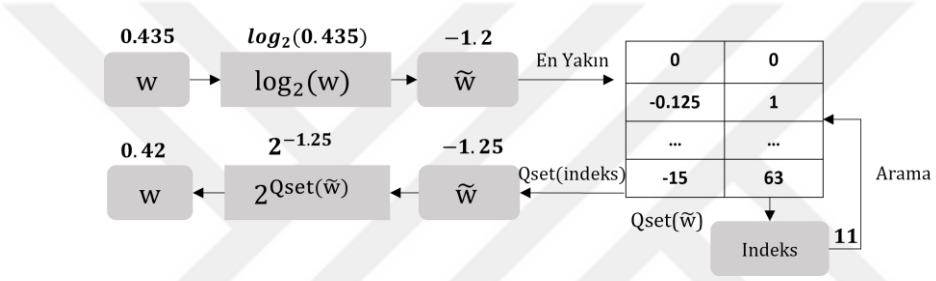
$$\text{P}_1 = \left\{0, -\frac{\text{R}}{2^{\text{n}}}, -\frac{2\text{R}}{2^{\text{n}}}, \dots, -\frac{\text{kR}}{2^{\text{n}}}\right\} \tag{5.10}$$

$$\text{P}_2 = \left\{-\left(\left\lfloor\frac{\text{kR}}{2^{\text{n}}}\right\rfloor + 1\right), -\left(\left\lfloor\frac{\text{kR}}{2^{\text{n}}}\right\rfloor + 1\right), \dots, -(\left\lfloor\frac{\text{kR}}{2^{\text{n}}}\right\rfloor + \text{M} - \text{k})\right\} \tag{5.11}$$

Burada $\text{M} = 2^{\text{n}} - 1$ ve $\text{k} = \left\lceil\frac{|\log_2(\text{S})|}{\text{R}} \cdot 2^{\text{n}}\right\rceil$.

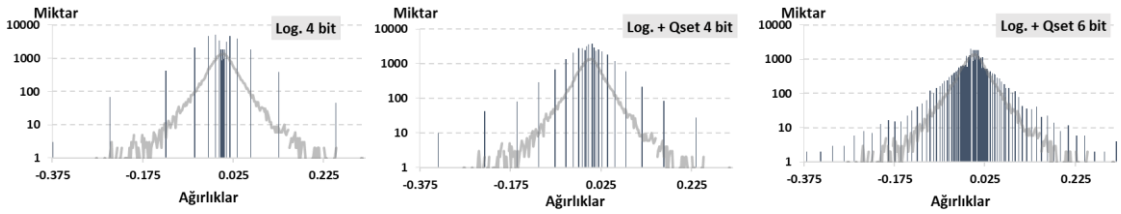
Qset kullanmanın arkasındaki fikir, gerçek sayı veya tamsayı kullanarak nicemlenmiş bir değeri temsil etmektir. S'den büyük mutlak ağırlıklar nicemleme seviyeleri olarak gerçek sayıları kullanır, aksi takdirde nicemleme seviyeleri olarak tam sayılar kullanılır. P1 kümesinde aralık, küçük bir adım boyutu kullanarak 0'dan, $-\frac{\text{kR}}{2^{\text{n}}}$ 'ye başlar. Bu şekilde, büyük değerli girdiler yüksek çözünürlükle nicemlenir. Ancak yalnızca küme P1'in kullanılması, küçük adım boyutu nedeniyle nicemleme aralığını sınırlar. Logaritmik şema zaten küçük değer girişleri için yüksek çözünürlük sağlamıştır. Bu nedenle nicemlenmiş değeri gerçek sayılar kullanarak sunmak gerekli değildir. Ayrıca daha küçük mutlak ağırlıkların doğruluk üzerinde daha büyük olanlardan daha az etkisi vardır. Bu nedenle P2'deki öğenin P1 kümesinin son öğesinden başlatıldığı P2 kümesinde bir tamsayı dizisi kullanılır. Bu şekilde, nicemleme aralığı daha fazla nicemleme seviyesini kapsayacak şekilde artırılır. Küme P2'yi kullanmak, basit logaritmik nicemleme yönteminde tam sayılara yuvarlamakla aynı şekilde çalışır.

S parametresi, P1 kümesinin aralığını tanımlamak için kullanılır. Deneylerinde, 1 ila 0.01 arasındaki mutlak ağırlıklar için yüksek çözünürlüklü nicemleme seviyeleri sağlamak üzere S, 0.01'e ayarlanmıştır. R parametresi pozitif bir gerçek sayıdır ve temsil seviyelerinin adım boyutunu ayarlamak için kullanılır. Büyük R, daha geniş bir nicemleme aralığı sağlar, ancak çözünürlük azalacaktır. P2 oluşturmak için, R parametresi $|\log_2 S|$'den büyük olmalıdır. Qset'in yapısını göstermek için R, n ve S parametreleri sırasıyla 8, 6 ve 0.01'e ayarlanır. P1 seti {0, -0,125, -0,25, …, -6,75} ve p2 seti {-7, -8, -9, …, -15} olarak verilebilir. Bu şekilde, Qset $[2^0, 2^{-15}]$ nicemleme aralığını sağlayabilir. Şekil 5.2'de önerilen logaritmik kuantizasyonun nicemleme süreci gösterilmiştir.



Şekil 5.2. Yüksek çözünürlüklü logaritmik nicemleme yöntemine genel bakış

Önerilen kuantizasyonun çözünürlüğü ve 4 bit kullanıldığında basit bir logaritmik nicemleme ile karşılaştırması Şekil 5.3'te gösterilmektedir. Sonuçlar, girişler sıfıra yakın olduğunda basit logaritmik yöntemin yüksek çözünürlüğe sahip olduğunu, ancak girdileri arttırırken çözünürlüğün önemli ölçüde düşük olduğunu göstermektedir.



Şekil 5.3. Önerilen logaritmik kuantizasyonun çözünürlüğü

Önerilen nicemleme, tüm girdi boyunca optimum çözünürlük sunmaktadır. Yüksek değerli girdiler için adım boyutu hala büyük olmasına rağmen, basit logaritmik yöntemin çözünürlüğünden çok daha iyidir. Ayrıca önerilen algoritma logaritmik nicemlemenin

sınırlı çözünürlüğünü ele almaktadır. Nicemleme için bit genişliğinin arttırılmasının çözünürlüğü daha da iyileştirebileceği görülebilir.

5.4.2. Özellik Haritaları İçin Nicemleme Yöntemi

Qset kullanılarak önerilen logaritmik nicemleme yöntemi, nicemleme çözünürlüğünde bir gelişme sağlamıştır. Bununla birlikte, özellik haritaları için Qset ile nicemleme yönteminin kullanılması, özellik haritaları ileri besleme işlemi sırasında kararlı olmadığı için standardın altında performansa yol açabilir. Bu nedenle, R parametresi Qset aralığı için tanımlanamaz. Ek olarak, Qset, birden küçük sayılar için nicemleme aralığı sağlar, böylece Qset, birden büyük değer alabilen özellik haritalarına uygulanamaz.

Yukarıdaki sorunu çözmek için P2 kümesi, pozitif sayı dizisiyle değiştirilir. R'nin nicemleme aralığı kaldırılır ve n'nin bit genişliği nicemleme aralıklarını sınırlamak için kullanılır. Burada adım boyutu, kesrin bit genişliği ile tanımlanır. Başka bir deyişle, nicemleme seviyesini temsil etmek için bir sabit nokta formatı kullanılır. Böylece adım boyutu, basit bir logaritmik nicemleme olarak bir tamsayı formatı kullanmaya kıyasla daha küçük olur. Bu nedenle ConNN modelinde çıktı özellik haritaları için önerilen Qset Denklem (5.12)'deki gibi gösterilebilir.

$$\text{Qset} = \{\{P_1\}, \{P_2\}\} \tag{5.12}$$

Burada

$$P_1 = \left\{0, -\frac{1}{2^{FL}}, -\frac{2}{2^{FL}}, \dots, -\frac{k}{2^{FL}}\right\} \tag{5.13}$$

$$P_2 = \left\{\frac{1}{2^{FL}}, \frac{2}{2^{FL}}, \dots, \frac{M-k}{2^{FL}}\right\} \tag{5.14}$$

Burada $M = 2^n - 1$ ve $k = 2^{n-1}$dır.

P1 kümesi, IL'nin tamsayı kısmının bit genişliği olduğu -2^{IL-1} 'ye basitleştirilebilen 0 ile $-\frac{k}{2^{FL}}$ arasındaki değerlerden oluşur. P1 kümesi, değeri 1'den küçük olan girdiyi nicemlemek için kullanılır. P2'deki nicemlenmiş değerler, 1'den büyük girdileri nicemlemek için pozitif sıra numaralarıdır. Kesir bitlerinin sayısı, adım boyutu değerini ayarlamak için kullanılır. Artan kesir bitleri, küçük bir adım boyutu sağlayabilir. Bu da

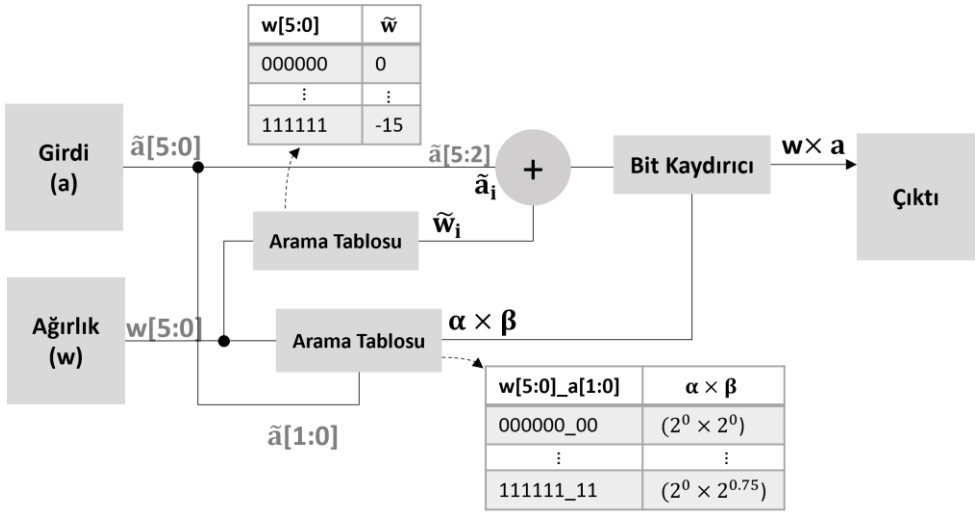
daha dar nicemleme aralıklarıyla yüksek çözünürlük olarak sonuçlanır. Bu iki küme, FL 0 ise basit bir logaritmik şema ile aynı nicemleme seviyelerini sağlar. Qset yapısını göstermek için kesrin bit genişliğinin 2 bit olduğu nicemleme için n = 6 bit olsun, o zaman P1 = {0, -0,25, -0,5, -0,75, …, -8} ve P2 = {0,25, 0,5, 0,75, …, 7,75} olarak ifade edilebilmektedir.

5.5. ConNN hızlandırıcı

Önceki bölümde, çarpmak-biriktirmek işlemi için PE'leri kullanan ConNN hızlandırıcısı uygulanmıştır. PE bloğu, bir küme çarpan ve toplayıcıdan oluşur. Bu kısımda, nicemlenmiş çıktının taban-2 değerinde bir güç kullanılarak temsil edildiği logaritmik nicemleme avantajından yararlanılmıştır. Bu nedenle, logaritmik ölçek sayıları için çarpma işlemi, çarpan yerine bit kaydırıcı ile gerçekleştirilebilir. Bunu göstermek için w ve a evrişimsel katmanındaki ağırlık ve çıktı özellik haritaları olsun. Logaritmik temsilde w ve a'nın çarpımı Denklem (5.15)'teki gibi ifade edilir.

$$
\begin{aligned}
& w \times a = Q_{slog}(w) \times Q_{slog}(a) \\
& = 2^{\tilde{w}} \times 2^{\tilde{a}} \\
& = 2^{\tilde{w}_i + \tilde{w}_f} \times 2^{\tilde{a}_i + \tilde{a}_f} \\
& = (2^{\tilde{w}_f} \times 2^{\tilde{a}_f}) \times 2^{\tilde{w}_i + \tilde{a}_i} \\
& = \text{Bit\_kaydırıcı}(\text{LUTs}(\alpha \times \beta), \tilde{w}_i + \tilde{a}_i)
\end{aligned}
\tag{5.15}
$$

Burada $\alpha = 2^{\tilde{w}_f}$, $\beta = 2^{\tilde{a}_f}$, $\tilde{a}_i$ ve $\tilde{a}_f$ nicemlenmiş çıktı özellik haritalarını tamsayı ve kesirli değerlerini gösterir. $\tilde{w}_i$ ve $\tilde{w}_f$, nicemlenmiş ağırlığın tamsayı ve kesirli değerlerini belirtir. Bit_kaydırıcı (), bit kaydırma işlevini belirtir ve Bit_kaydırıcı (x, y) x'i y bit kaydırmayı gösterir. LUTs (), bir arama tablosunu belirtir ve LUTs (k), k'nin bir arama tablosu girişleri anlamına gelir. 5.15 denkleminde, w ve a'nın çarpımı bit kaydırma işlemi kullanılarak gerçekleştirilebilir. Ancak nicemleme düzeylerini sunmak için gerçek bir sayı kullandığımızdan, nicemlenmiş sonuç tam bir tam sayı değildir. Kesir değerleri için bit kaydırma işlevi kullanılamaz. Kesir değerlerinin çarpılmasını önlemek için $\alpha \times \beta$ sonucunu 2^N girişini içeren arama tablosunda saklanır. Burada N, $\tilde{w}_f$ ve $\tilde{a}_f$ 'nin bit genişliğidir. Şekil 5.4 model 6 bit olarak nicemlenirken evrişimin hesaplanması için hızlandırıcının örnek mimarisini göstermektedir.



Şekil 5.4. Bit kaydırıcı kullanan hızlandırıcının donanım yapısı

Hızlandırıcının mimarisi, bit kaydırıcılar, toplayıcılar ve arama tablolarından oluşur. Hafızada saklanan ağırlık Qset indeksini gösterdiğinden, toplayıcıya girmeden önce indeksi nicemlenmiş ağırlıkları ($\tilde{w}_i$) tamsayı kısmına eşlemek için arama tablosu kullanılır. $\alpha \times \beta$ hesaplaması için çarpma işlemi tüm çarpma sonuçlarını saklayan arama tablosu ile değiştirilir. Bu arama tablosu oluşturulur ve çarpma sonucunu eşleştirmek için ağırlık indeksi ile nicemlenmiş özellik haritası (a[1:0]) birleştirilir. Bu arama tablosunda, her bir öğenin $\alpha \times \beta$ sonucuna eşlendiği $2^{(6+2)}$ öğe bulunur. Önerilen hızlandırıcı, iki arama tablosu için ek bellek arabellekleri gerektirmesine rağmen, FPGA'da maliyetli ve sınırlı DSP bloklarının kullanımını önemli ölçüde azaltır.

5.6. Deneysel Sonuçlar

5.6.1. Nicemleme Doğruluğu

Önerilen nicemleme yönteminin performansı, AlexNet, ResNet-18 ve YOLOv3-tiny dahil olmak üzere farklı ConNN modelleri kullanılarak sunulmuştur. ImageNet veri kümesinde bir görüntü sınıflandırması yapılırken AlexNet ve ResNet-18 modellerinin ölçümü için en yüksek doğruluk oranı (Top-1) hesaplanmıştır. YOLOv3-tiny için COCO veri kümesinde nesne algılama gerçekleştirirken modelin doğruluğunu değerlendirmek için genel ortalama kesinlik (mAP) kullanılmıştır.

Yöntemimizi kullanırken doğruluktaki gelişmeyi gözlemlemek için basit logaritmik şema ile önerilen yüksek çözünürlüklü logaritmik yöntem arasında bir karşılaştırma Tablo

5.1'de gösterilmiştir. W ve A bitleri, ağırlıkların bit genişliğini ve özellik haritaları tanımlar. $Q_{\log}$ ve $\mathrm{Q}_{\mathrm{slog}}$ sırasıyla basit bir logaritmik ve yüksek çözünürlüklü nicemleme yöntemlerini belirtir. FP, modelin orijinal doğruluk oranını gösterir.

Tablo 5.1. Yüksek çözünürlüklü logaritmik nicemleme performansı

Model	W/A bit	$\mathrm{Q}_{\mathrm{log}}(\mathrm{w},\mathrm{a})$	$\mathrm{Q}_{\mathrm{slog}}(\mathrm{w},\mathrm{a})$
AlexNet FP: 57,2	8/8	44,7	56,68
	6/6	44,7	56,25
	5/5	42,91	54,67
	4/4	39,5	53,2
ResNet-18 FP: 70,7	8/8	36,29	69,54
	6/6	36,29	69,23
	5/5	36,2	67,91
	4/4	26,04	59,94
YOLOv3-tiny FP: 24,5	8/8	19,23	23,92
	6/6	19,23	23,79
	5/5	19,11	22,5
	4/4	17,45	20,5

Sonuçlar, önerilen $\mathrm{Q}_{\mathrm{slog}}$ yönteminin tüm bit genişlikleri için basit logaritmik nicemleme yönteminden daha iyi performans sergilediğini göstermiştir. 8-bit nicemleme ile önerilen yöntem, orijinal doğruluğa yakın doğruluk sonuçları elde etmiştir. Bu sonuç, aynı zamanda, bit genişliği sayısı artırılırken, yüksek çözünürlük ölçeklerinin kullanılmasının iyileştirmeler sağladığını da göstermiştir. Örneğin, $\mathrm{Q}_{\mathrm{log}}$'da 4 bitten büyük bit genişlik, çözünürlük değişmediğinden doğruluk performansını iyileştirmemiştir. Bunun yanı sıra önerilen nicemleme yöntemi bit genişliğini artırarak doğruluğu artırabilmiştir. Tablo 5.2'de performansların karşılaştırılması gösterilmektedir.

Karşılaştırma sonuçları, önerilen yöntem, yüksek çözünürlüklü bir adım boyutu ile logaritmik bir nicemleme kullanan (Huang ve diğ., 2021) çalışmasından daha iyi performans sunduğunu göstermiştir. Önerilen tasarım, (Banner ve diğ., 2018) ile benzer bir doğruluk sağlamıştır. Çalışmaları, nicemleme için analitik bir kırpma yöntemi ve her kanal için en uygun bit genişliğini belirlemeye yönelik bir teknik kullanarak eğitim sonrası nicemleme sunmuştur. (Oh ve diğ., 2021) ve (T. Y. Lu ve diğ., 2020) ile karşılaştırıldığında, sonuçların önerilen yöntemin ağırlıkları 5 bitin altına nicelleştirildiğinde çok iyi performans sunmadığı gözlenmiştir.

Tablo 5.2. ResNet-18'i değerlendiren önceki çalışmalarla karşılaştırma

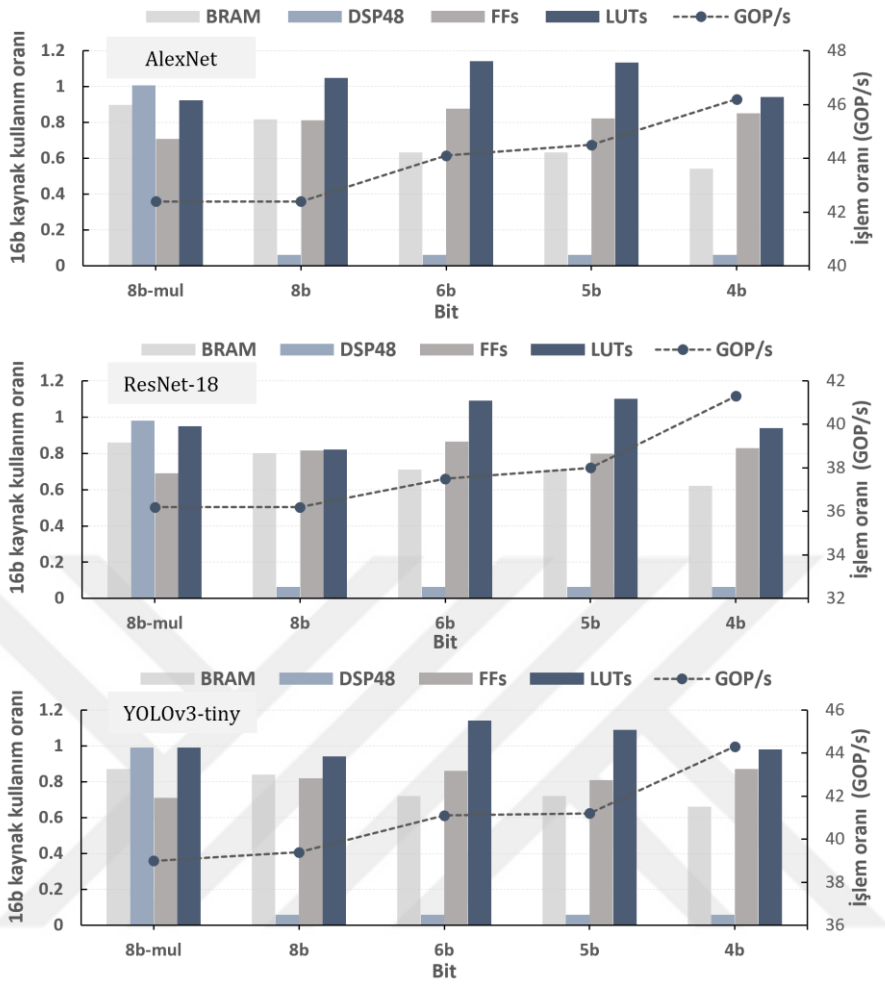
Yöntem	W/A bit	İlk 1 doğruluk oranı	Yeniden eğitim
(Huang ve diğ., 2021)	8/8	67,61	×
Önerilen yöntem		69,54	×
(Banner ve diğ., 2018)	8/4	66,6	×
Önerilen yöntem		69,79	×
(T. Y. Lu ve diğ., 2020)	4/8	69,18	✓
Önerilen yöntem		63,12	×
(Oh ve diğ., 2021)	5/5	69,72	✓
Önerilen yöntem		67,91	×

5.6.2. Donanım Kullanım Sonuçları

Nicemlenmiş ConNN modelleri çalıştırmak için donanım hızlandırıcı, XC7Z020 FPGA cihazında uygulanmıştır. Döşeme ve açma değişkenleri gibi donanım konfigürasyonları, Bölüm 3 ve 4'teki önceki tasarımla aynı konfigürasyonla ayarlanmıştır. Hızlandırıcının kaynak kullanımı, 16 bit tabanlı ConNN uygulamasıyla karşılaştırıldığında ConNN modellerinde nicemleme yöntemini kullanarak Şekil 5.5'te rapor edilmiştir.

Öncelikle, FPGA'da kaynak gereksinimlerini gözlemlemek için basit çarpmalar (8b mul) kullanarak 8-bit ConNN modeli için hızlandırıcı uygulanmıştır. Ardından, bit kaydırma operatörleri kullanılarak önerilen logaritmik hızlandırıcı sunulmuştur. Sonuçlar, veri hassasiyetini 8 bite düşürmenin 16 bite kıyasla daha az kaynak tükettiğini göstermiştir. Bit kaydırıcı kullanan 8 bit hızlandırıcının DSP kullanım sayısı, 16 bit hızlandırıcıya kıyasla önemli ölçüde azaltılmıştır. Ancak, 8 ve 6-bit hızlandırıcılar, 16-bit hızlandırıcıdan daha fazla LUT kaynağı kullanmıştır. Hızlandırıcıdaki LUT'lerin ek yükü, bit kaydırma işlemlerini gerçekleştirmek için kullanılmıştır. Bit genişliği 4 bite düşürüldüğünde kaynak kullanım sayısı azalmıştır.

Sonuçlar, tüm kaynak kullanım sonuçlarında 4 bit kullanıldığında kaynak kullanım oranının 16 bitten daha düşük olduğunu göstermiştir. İşlem oranı, bit genişliğini azaltırken iyileşme eğilimindedir. Deneysel sonuçlar, bit genişliğini 4 bite düşürmenin, sırasıyla 8 ve 16-bit ile karşılaştırıldığında yaklaşık 1,13× ve 1,2× daha yüksek işlem oranı hızı sağladığını göstermiştir. Ayrıca önerilen tasarım kaynak verimliliği metrikleri ile değerlendirilmiştir.



Şekil 5.5. Logaritmik nicemlenmiş ConNN modelleri için hızlandırıcının kaynak kullanım oranı ve işlem oranı performansı

Tablo 5.3'te ve Tablo 5.4'te hızlandırıcıyı farklı bit genişliği için dağıtırken kaynak verimliliği sonuçlarını gösterilmektedir. DSP verimliliğinin sonuçları, logaritmik hızlandırıcının 16-bit hızlandırıcıdan önemli ölçüde daha iyi performans gösterdiğini göstermiştir. 4-bit hızlandırıcı, diğer bit genişlikli tasarımlara kıyasla en yüksek DSP verimliliğine ulaşmıştır. 4-bit genişlikli uygulama sonuçlarında hızlandırıcı, sırasıyla AlexNet, ResNet-18 ve YOLOv3-tiny üzerinde değerlendirirken 16 bit tabanlı tasarıma göre 19,3×, 17,2× ve 18,4× daha yüksek DSP verimliliği sağlamıştır.

Tersine, logaritmik hızlandırıcı, 16-bit hızlandırıcıya kıyasla LUT verimliliğinde hafif bir gelişme sağlamıştır. 8, 6 ve 5 bit için önerilen hızlandırıcı, AlexNet sonucunda gösterildiği gibi çarpanları kullanan 8-bit hızlandırıcıdan daha düşük LUT verimliliği elde etmiştir. ResNet-18 ve YOLOv3-tiny için 8 bit logaritmik hızlandırıcı daha iyi LUT

verimliliği sağlarken, 6 ve 5 bit gibi daha düşük bit genişliği, çarpanları kullanan 8 bit hızlandırıcıya kıyasla daha düşük performans sağlamıştır. Ancak tüm modellerin sonuçları, hızlandırıcıyı 4 bit ile dağıtmanın diğer bit genişlikli tasarımlara kıyasla daha yüksek LUT verimliliği sağlayabileceğini ortaya koymuştur.

Tablo 5.3. Önerilen nicemleme kullanılırken DSP verimliliği

Model	GOP/s/DSP					
	16b	8b-mul	8b	6b	5b	4b
AlexNet	0,3	0,33	5,2	5,5	5,6	5,8
ResNet-18	0,26	0,29	4,53	4,7	4,75	5,16
YOLOv3-tiny	0,28	0,31	4,93	5,14	5,15	5,54

Tablo 5.4. Önerilen nicemleme kullanılırken LUT verimliliği

Model	GOP/s/kLUT					
	16b	8b-mul	8b	6b	5b	4b
AlexNet	2,5	3,01	2,62	2,53	2,57	3,24
ResNet-18	2,45	2,77	3,21	2,5	2,52	3,21
YOLOv3-tiny	2,55	2,81	2,95	2,53	2,65	3,18

5.7. Sonuçlar

Bu bölüm, ConNN modellerinin hesaplama karmaşıklığını basitleştirmek için bir logaritmik şema kullanan bir eğitim sonrası nicemleme yöntemini göstermiştir. Bu çalışmada optimal bir logaritmik ölçek kullanılarak logaritmik nicemlemeyi geliştirmek için basit ama etkili bir yöntem önerilmiştir. 2'nin gücünün avantajıyla, çarpanların bit kaydırıcılar, toplayıcılar ve arama tabloları ile değiştirildiği ConNN çıkarımı için bir donanım hızlandırıcı sunulmuştur. Değerlendirme sonuçları, logaritmik nicemlenmiş modellerin, yeniden eğitim olmaksızın kayan-nokta modellere yakın doğruluk elde ettiğini göstermektedir. Donanım değerlendirme sonuçlarında, hızlandırıcı hem çıktıyı hem de kaynak verimliliğini arttırmaktadır. Çarpan, konvolüsyon hesaplaması için gerekli olmadığından özellikle DSP bloğu olmak üzere kaynak gereksinimlerinin sayısı önemli ölçüde azalmaktadır.

6. AYKIRI AĞIRLIKLARIN FARKINDA NİCELEME YÖNTEMİ

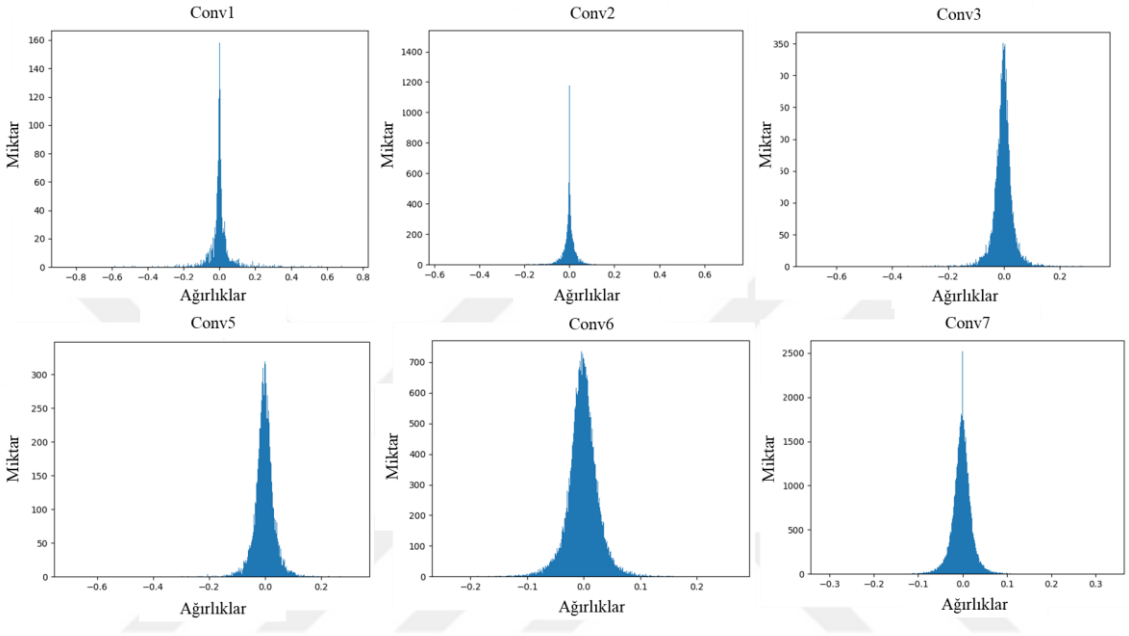
Önceki bölümde, ConNN sıkıştırması için yüksek çözünürlüklü logaritmik nicemleme yöntemi sunulmuştur. Önerilen yöntemi, basit bir logaritmik şemaya kıyasla doğruluğu önemli ölçüde artırmıştır. Ayrıca çarpansız hızlandırıcı ile evrişim hesabı yapılabilen donanım dostu bir tasarım sunmuştur. Ancak önerilen logaritmik nicemleme kullanmanın dezavantajları vardır. Hızlandırıcıyı FPGA'da uygulamak için kesirli parçaların çarpma sonuçlarını tutmak ve arama tablolarını dağıtmak için fazladan iç bellek gerekir. Doğruluk sonuçları için çok düşük bit genişliğine (8 bitin altında) nicemleme yapılırken önerilen nicemlemenin iyi gerçekleşmediği gözlenmiştir. Bu sebeple, bu bölüm, önceki önerilen yönteminin performansını iyileştirmeye yönelik teknikler sunmaktadır.

6.1. Tanıtım

ConNN modelleri üzerinde çalışılan birkaç çalışma, ConNN'deki ağırlık parametrelerinin çan eğrisi ve uzun kuyruk dağılımı takip ettiğini göstermiştir. Başka bir deyişle, çok sayıda ağırlığın değerleri sıfır civarındadır ve çok azı yüksek mutlak değerlere sahiptir. Ağırlıklar üzerinde logaritmik nicemlemenin kullanılması, logaritmik yöntemin sıfır civarında yüksek bir çözünürlüğe sahip olması nedeniyle iyi bir performans sağlamıştır. Giriş sıfıra yakın olduğunda basit logaritmik yöntem iyi çalışmıştır. Ancak girdi sıfırdan uzak olduğunda nicemleme performansı azalmıştır. Önceki bölümde, bu problem yüksek çözünürlüklü logaritmik ölçekler kullanılarak çözülmüştür. Performansı daha da iyileştirmek için nicemleme stratejisine odaklanmak yerine bu bölümde ağırlık yapısı üzerinde çalışılmıştır. Şekil 6.1'de ResNet-18'in bazı katmanlarındaki ağırlıkların dağılımı gösterilmektedir.

Ağırlık dağılım şekli, ağırlıkların çoğunun sıfıra yakın küçük değerler olduğunu göstermektedir. Ancak bazı katmanlardaki ağırlık yapısı, büyük bir mutlak değere sahip bir dizi ağırlık olduğu anlamına gelen uzun bir dağılım kuyruğuna sahiptir. Kuyruktaki bu yüksek değerli ağırlıklar, çıktı özellik haritalarıyla güçlü bir bağlantıya sahip oldukları için ConNN modelinin doğruluğunda önemli değişikliklere yol açabilmiştir. Basitçe bit sayısını artırmak daha geniş ve yüksek çözünürlüklü bir nicemleme yöntemi sağlayarak bu sorunu çözebilse de donanım gereksinimlerinin bir ek yükü vardır. Ayrıca aynı sayıda bit genişliğinin tüm ConNN modelleri için çalışabileceğini garanti etmemiştir. Uzun

kuyruklu dağıtım sorununu ele almak için aykırı değer ayrımı adı verilen bir ağırlık dağılımını yeniden şekillendirme yöntemi önerilmiştir. Ağırlık dağılımının kuyruğunda yer alan büyük mutlak ağırlık, aykırı değer olarak kabul edilmiştir. Bu yöntem, ağırlıkların aykırı değerini iki küçük ağırlığa böler. Ağırlıkların yeni bir dağılım şekli, nicemleme için daha uygun bir girdi haline gelir.



Şekil 6.1. ResNet-18'de ağırlık dağılımı örneği

6.2. İlgili Çalışmalar

Çok düşük bit genişliğine sahip olan ConNN modellerinin nicemleme işlemi sınırlı nicemleme seviyelerine sahiptir ve bu yüzden ağırlık veya özellik haritası büyük değerlere sahip olduğunda iyi performans göstermeyebilir. Bu sorunu çözmek için nicemlemenin bir girdisini inceleyen birkaç çalışma vardır. (Park ve diğ., 2018) aynı dağılımdaki büyük mutlak veriler için yüksek bit genişlikli (16 veya 32 bit) bir nicemleme kullanılırken, küçük mutlak veriler için düşük bit genişlikli bir nicemleme uygulayan karma kesinlikli bir nicemleme uygulamıştır. Deneyleri, yeniden eğitimden sonra 4-bit nicemleme kullanıldığında yalnızca %1 doğruluk kaybı olduğunu göstermektedir.

(Yan ve diğ., 2020), ağırlıkların dağılımının üç parçaya bölündüğü ve daha sonra yöntemin bir parçayı nicemlemeye başlarken, diğer parçalar için nicemleme aralığını doğruluk düşüşüne göre ayarladığı çoklu faz uyarlaması adı verilen bir nicemleme süreci

sunmuştur. Önerilen teknikleri, önemli doğruluk kaybı olmadan ConNN modellerini 3 ve 4 bit genişliğinde nicemlemeyi başarmıştır. Ağırlıkların ve aktivasyonların dağılımının yeniden şekillendirilmesi önerilmiştir (R. Zhao ve diğ., 2019). Bu yöntem, aykırı değerleri ikiye bölmeyi kullanarak daha küçük iki değere bölmektedir. Bölmeden önce ağırlıklara önyargı eklemenin, ikiye bölmekten daha düşük bir nicemleme hatası sağlayabileceğini öne sürmüştür. Bu yöntem, ağırlıkların dağılımını çan şeklinde ve kısa kuyruklu bir dağılıma dönüştürmüştür.

6.3. Aykırı Ağırlık Ayrımı

6.3.1. Aykırıların Tanımlanması

İstatistik çalışmalarında, bir aykırı değer, aynı dağılımın diğer değerlerine kıyasla alışılmadık derecede büyük veya küçük bir değerdir. Aykırı değerler, çıktı üzerinde yüksek veya düşük etkisi olabilecek bir veri problemi olarak görülebilmiştir. Benzer şekilde, bir ConNN modelinin ağırlıkları, çoğunlukla sıfıra yakın olan bir değerler kümesidir. Ancak diğer ağırlıklardan önemli ölçüde daha büyük veya daha küçük olan birkaç ağırlık vardır. ConNN modelindeki bu büyük ağırlıklar, doğruluk üzerinde daha küçük olanlardan daha büyük bir etkiye sahiptir.

Bu tezde, büyük ağırlıkların dağılımda aykırı değerler olduğu düşünülmüştür. Yüzdelik nokta, aykırı değerleri belirlemek için ağırlıklar sıralandıktan sonra metrik olarak kullanılmaktadır. Optimal yüzdelik noktasını seçmek için sunulan etkin bir yöntem yoktur. Ancak, ConNN modelleri için en uygun aykırı noktayı gözlemlemek için farklı yüzdelik noktaları kullanan nicel deney tanıtılmıştır.

6.3.2. Ağırlık Ayrımı

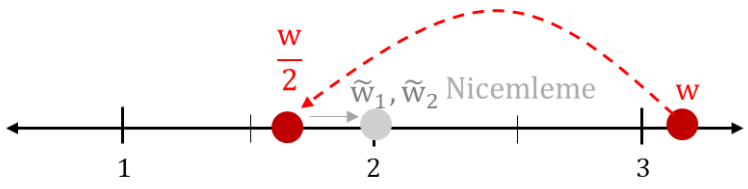
Ağırlık ayrımı, aykırı değer ağırlıklarının büyüklüğünü azaltmanın etkili yollarından biridir. (R. Zhao ve diğ., 2019), ağırlıkları ikiye bölerek ağırlıkların değerini azaltmak için bir teknik olan bir ağırlık bölme yöntemi önermiştir. Ağırlık ayırma yöntemini sunmak için, y_i, i'nci çıktı özellik haritası olsun, girdi özellik haritaları $x = \{x_j\}_{j=0}^{n}$ ve $w_{i,j}$, x_j ve y_i 'yi bağlayan ağırlıkları temsil etsin. Ağırlık ayırma yöntemi benimsenirken i. çıktı özelliği haritasının (y_i) hesaplanması Denklem (6.1)'deki gibi ifade edilebilir.

$$y_i = \sum_{j=1}^{n} x_j w_{i,j}$$
$$= \sum_{j=1}^{n-1} x_j w_{i,j} + x_n\left(\frac{w_{i,n}}{2} + \frac{w_{i,n}}{2}\right)$$
$$= \sum_{j=1}^{n-1} x_j w_{i,j} + \left(x_n \frac{w_{i,n}}{2}\right) + \left(x_n \frac{w_{i,n}}{2}\right) \quad (6.1)$$

Ağırlık $w_{i,n}$ yarıya indirilmiş ve girdi özellik haritası x_n, yarıya bölünmüş ağırlıklarla çarpma yapmak için çoğaltılmıştır. Bu nedenle, ağırlıkların değeri yarı yarıya azalmıştır. Bu yöntem, etkilenen ağırlıkları ortalamaya doğru hareket ettirerek ağırlık dağılımını yeniden şekillendirmiştir. Kuyruklu dağılımdaki aykırı değerler kaldırılmış ve dağılımın zirvesi daha da yükselmiştir. Bu, logaritmik yöntemin ortalamada yüksek sayıda nicemleme düzeyine sahip olması nedeniyle, logaritmik nicemlemenin daha iyi çalışmasını teşvik etmiştir. Bu nedenle, önerilen logaritmik nicemleme kolaylaştırmak için bu ağırlık ayırma tekniği benimsenmiştir.

6.3.3. Kalıntı Kullanarak Aykırı Ağırlık Ayrımı

Nicemleme yöntemi, orijinal bir değerden yaklaşık bir değer üretmektedir. Nicemleme işlemi sırasında oluşan bir hata vardır. Başka bir deyişle, nicemlenmiş bir değer, ilave gürültü içeren orijinal bir değerdir. Bu nedenle, tam ağırlıkların nicemlenmesi $Q(w), Q\left(\frac{w}{2}\right) + Q(\frac{w}{2})$ ile aynı sonucu vermemiştir. Ayrıca niceleyici aynı yönde yuvarlandığında, iki yarıya bölünmüş ağırlığın nicemlenmesinden kaynaklanan ek gürültü iki katına çıkabilmektedir. Girdi (w) 3,1 olduğunda bölme yöntemini kullanan bir nicemleme örneği Şekil 6.2'de gösterilmektedir.



Şekil 6.2. Ağırlık bölme yöntemini kullanan bir nicemleme örneği

Tam girdi kullanılması durumunda, nicemlemenin sonucu ($\tilde{w}$) 3'tür. Yarıya bölünmüş girdi ($\frac{w}{2} = 1.55$) kullanılması durumunda, nicemlemenin sonucu ($\tilde{w}_1$ ve $\tilde{w}_2$) 2'dir. Bu nedenle, nicemlemenin iki çıktısı $\tilde{w}_1$ + $\tilde{w}_2$ = 4'tür. Tam giriş için nicemleme hatası 3.1-3 = 0.1 iken, yarıya bölünmüş girdi için nicemleme hatası 4-3.1 = 0.9'dur.

Doğrusal nicemleme için, (R. Zhao ve diğ., 2019), ikiye bölmeden önce ağırlıklara ± 0.5 ekleyerek bu genel gürültü sorununu ele almıştır. İlk yarıya bölünmüş ağırlık $w_1 = \frac{w-0.5}{2}$'tir. Diğer bir yarıya bölünmüş ağırlık $w_1 = \frac{w+0.5}{2}$'tir. Bu bölme yöntemi, w iki tamsayı noktası arasındaki orta noktaya yakın olduğunda Q(w)'yi farklı yönlerde yuvarlamaya zorlamıştır. Bu şekilde, iki yarıya bölünmüş ağırlığın nicemlemesi, orijinal ağırlığın nicemlemesi ile aynı sonuçları vermiştir.

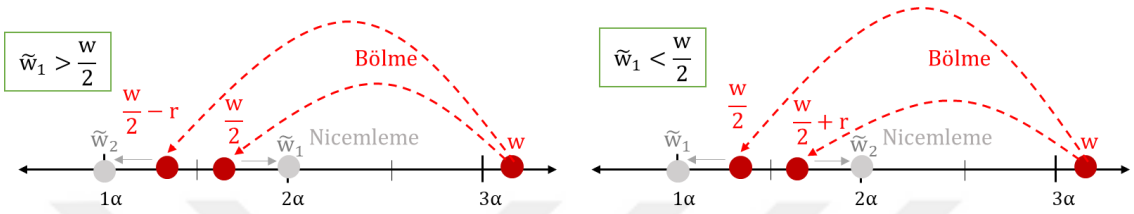
Ancak ağırlıkların ikiye bölme kullanılarak ayrılması ve sabit yanlılığın eklenmesi, tek biçimli nicemleme özelliklerinden dolayı logaritmik nicemleme için alt-optimal performansa yol açabilmektedir. Logaritmik şemanın adım boyutları, aralık boyunca eşit değildir. Örneğin, 2^{-1} ve 2^{-2} nicemleme seviyesi arasındaki adım boyutları 0,25'tir. Ancak adım boyutu 2^{-3} ve 2^{-4} arasında 0,0625'tir. Bu sorunu ele almak için kalıntı kullanılarak bir aykırı ağırlık ayrımı önerilmiştir. Önerilen fonksiyon Denklem (6.2)'de gibi ifade edilebilir.

$$\begin{aligned} \mathrm{w} \rightarrow (\mathrm{w}_1, \mathrm{w}_2) &= \left(\frac{\mathrm{w}}{2}, \frac{\mathrm{w}}{2} + (\frac{\mathrm{w}}{2} - \tilde{\mathrm{w}}_1)\right) \\ &= \left(\frac{\mathrm{w}}{2}, \mathrm{w} - \tilde{\mathrm{w}}_1)\right) \end{aligned} \tag{6.2}$$

Burada $\tilde{\mathrm{w}}_1 = \mathrm{Q}(\frac{\mathrm{w}}{2})$.

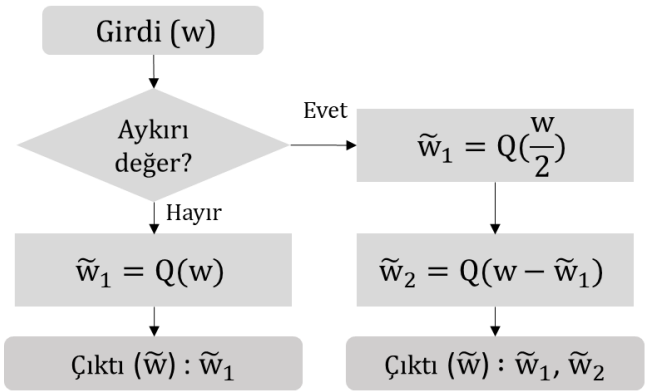
Bu fonksiyonun kavramı, ağırlık ayrımının yönünü belirlemek için ilk ağırlıkların nicemlemesinden kaynaklanan hatayı kullanmıştır. İlk ayrı ağırlık (w_1) orijinal ağırlığın ikiye bölünmesidir. İkinci ayrı ağırlık (w_2) için $\tilde{\mathrm{w}}_1$'in nicemleme sonucunu telafi etmek için yarıya bölünmüş ağırlığa $\frac{\mathrm{w}}{2} - \tilde{\mathrm{w}}_1$ kalıntısını eklemiştir. Bu nedenle, $\mathrm{w}_2, \mathrm{Q}(\frac{\mathrm{w}}{2})$'nin nicemleme hatasına bağlıdır. w_2 ağırlığı $\frac{\mathrm{w}}{2} - \tilde{\mathrm{w}}_1$ olduğunda $\frac{\mathrm{w}}{2}$'den daha büyüktür ve w_2 ağırlıkları $\tilde{\mathrm{w}}_1 > \frac{\mathrm{w}}{2}$ olduğunda $\frac{\mathrm{w}}{2}$'den daha küçüktür.

Nicemleme hatası sıfır ise ($\frac{w}{2} - \tilde{w}_1 = 0$), $w_2 = \frac{w}{2}$ olur. $Q(w_1)$ ve $Q(w_2)$, $Q(w_1)$'nin nicemleme hatası, iki bitişik nicemleme seviyesinin mesafesinin yarısından az olduğunda, aynı yöne doğru yuvarlanır. Aksi durumda ise, $Q(w_1)$ ve $Q(w_2)$ farklı yönlere yuvarlanır. Bu yaklaşım ayrıca w_1 iki nicemleme seviyesi arasındaki orta noktaya yakınsa, aynı yöne yuvarlama sırasında oluşan çift hatayı da önler. Önerilen nicemleme yönteminin bir özeti, Şekil 6.3'de gösterilmiştir.



Şekil 6.3. Önerilen ağırlık ayırma yöntemine genel bakış

Örneğin, nicemlemenin girdisi 3.1 olsun. Önerilen yöntem, girdiyi ikiye böler ve ardından bölünmüş girdi üzerinde nicemleme gerçekleştirir. Dolayısıyla $\tilde{w}_1$, yarıya bölünmüş girdi kullanılarak nicemlenmiş çıktıdır. Bu nicemlemede $\tilde{w}_1$ 2'dir. Bu şekilde, artık değer $r = 1.55 - 2 = -0.45$'tir. Dolayısıyla başka bir girdi $\frac{w}{2} - r = 1.55 - 0.45 = 1.01$'dir, böylece nicemleme çıktısı $\tilde{w}_2 = 1$ olmuştur. Nicemlemenin iki çıktısı $\tilde{w}_1 + \tilde{w}_2$ = 3'tür. Tam giriş için nicemleme hatası 3.1-3 = 0.1 iken, yarıya bölünmüş girdi için nicemleme hatası 4-3.1 = 0.9'dur. Önerilen yöntem, tam girdi için nicemlemeye aynı çıktıyı sağlamıştır. Şekil 6.4'de hem tam ağırlıklar hem de aykırı ağırlıklar için önerilen nicemleme yönteminin diyagramı sunulmuştur.

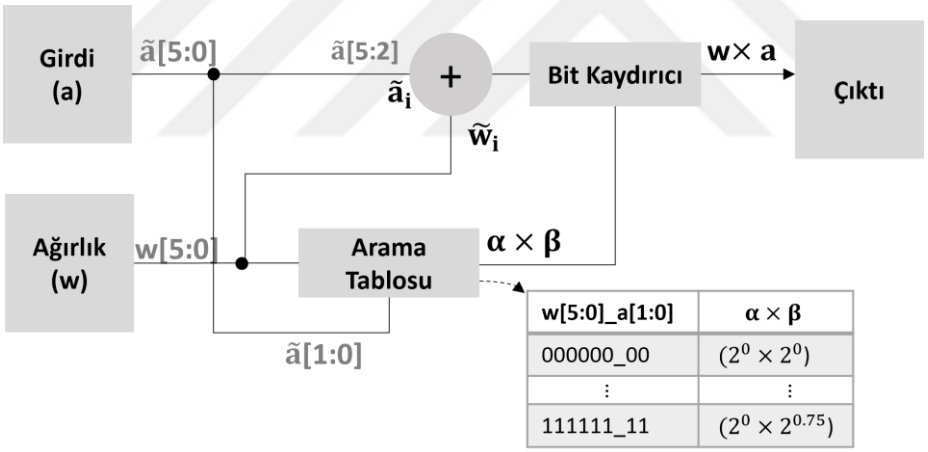


Şekil 6.4. Ağırlık ayırma yöntemi ile önerilen nicemleme işlemi

Ağırlık bir aykırı ağırlık ise, bu yöntem önce ağırlık ikiye böler ve ardından bu bölünmüş değer ile nicemleme gerçekleştirir. İkinci nicemlenmiş ağırlık elde etmek için orijinal ağırlık, birinci ağırlığın nicemlenmiş değeriyle çıkarılmış ve ardından nicemleme, bir girdi olarak çıkarma sonucu ile gerçekleştirilir. Önerilen ayırma tekniği, doğrusal ve logaritmik nicemleme algoritmaları için optimal değerler sağlamıştır.

6.4. Donanım Hızlandırıcı

Önceki bölümde sunulan çarpansız donanım hızlandırıcı, çarpma sonuçlarını ve nicemlenmiş ağırlıkların gerçek değerlerini depolamak için iki arama tablosu gerektirmiştir. Bu bölümde hem ağırlıklar hem de çıktı özellik haritaları için kesir bitlerine dayalı Qset ile logaritmik nicemleme yöntemi benimsemiştir. Bu nedenle, nicemlenmiş ağırlıkların gerçek değerlerini depolamak için kullanılan arama tablosu kaldırılabilir. Bit kaydırıcı kullanan ConNN hızlandırıcının donanım yapısı Şekil 6.5'te gösterilmiştir. Bu donanım mimarisi 6-bit hızlandırıcı için tasarlanmıştır.



Şekil 6.5. Logaritmik nicemlenmiş ConNN modeli için donanım hızlandırıcı

Önceki tasarıma benzer şekilde, hızlandırıcının mimarisi bit kaydırıcı operatörü, toplayıcı ve arama tablosundan oluşmuştur. Nicemlenmiş ağırlıkların bir Qset indeksi olarak saklanması gerekli değildir. Gerçek değerle nicemlenmiş ağırlığı eşleştirmek için bir arama tablosu gerekli değildir. Bu nedenle, hızlandırıcı, nicemlenmiş ağırlıkların ve özellik haritalarının ($\alpha \times \beta$) kesirli bölümlerinin çarpma sonuçlarını depolamak için bir arama tablosuna ihtiyaç duymuştur. w_1 ve w_2 kesir bitinin aynı boyutuna sahip

olduğundan, ayrılan ağırlıklar çarpma sonucunu eşlemek için aynı arama tablosunu kullanabilir.

6.5. Deneysel Sonuçlar

6.5.1. Nicemlenmiş Modellerin Doğruluk Sonuçları

ConNN modellerinde ağırlıklar ve çıktı özellik haritaları için yüksek çözünürlüklü logaritmik nicemleme yöntemi kullanılmıştır. Sadece ağırlıklar için aykırı ağırlık ayrımı yöntemi uygulanmıştır. İlk olarak, özellik haritalarının nicemlenmesinde bozulma etkisi olmaksızın önerilen çalışmanın performansını gözlemlemek için özellik haritaları 8-bit olarak nicemlenmiştir. Ağırlıklar için bit genişliği ise 8 ile 4-bit arasında değiştirilmiştir. Ayrıca, önerilen çalışmanın performansı, farklı yüzdelik noktası (P) kullanıldığında da göstermiştir. ImageNet veri kümesinde görüntü sınıflandırması için nicemlenmiş ResNet-18 modelinin deneysel sonuçları Tablo 6.1'de gösterilmiştir. Yarılama yöntemi, iki ile basit bir ağırlık bölümünü ifade etmiştir. Kanıtlı yöntemi, önerilen ağırlık ayırma tekniğini belirtmiştir. W, ağırlıkların bit genişliğini belirtmiştir.

Tablo 6.1. Nicemlenmiş ResNet-18 için elde edilen doğruluk oranları. Özellik haritasının bit genişliği 8-bit olarak sabitlenmiş ve ağırlık bitleri değiştirilmiştir

Yöntem	W-bit	P_{99}	P_{95}	P_{90}	P_{80}
Yarılama	8	69,3	69,4	69,5	69,2
+ Kalıntı		69,4	69,6	69,6	69,5
Yarılama	6	69,2	69,2	69,2	69,2
+ Kalıntı		69,3	69,5	69,4	69,4
Yarılama	4	63,2	63,4	63,4	63,5
+ Kalıntı		65,9	66,4	66,8	66,4

Önerilen yöntem, her bit genişliği nicemlemesinde basit yarıya bölme yönteminden daha iyi performans göstermiştir. 8-bit nicemleme için önerilen yöntem, yarılama yöntemine kıyasla doğrulukta hafif bir gelişme sağlamıştır. Ancak bu yöntem, modeli 4-bit olarak nicelleştirirken yarılama yönteminden önemli ölçüde daha yüksek doğruluk elde edebilmiştir. Bu deneysel sonuç, önerilen yöntemin ağırlık bölme yöntemi için optimal bir yöntem olabileceğini kanıtlamıştır.

Bit genişliğinin doğruluk üzerindeki etkisini incelemek için, önerilen nicemleme farklı bit genişlikleri kullanılarak değerlendirilmiştir. Aykırı değerlerin etkisini gözlemlemek ve ConNN modelinin optimal doğruluğunu araştırmak için farklı aykırı değer noktaları uygulanmıştır. Tablo 6.2'de AlexNet, ResNet-18 ve YOLOv3-tiny üzerinde değerlendirilen önerilen çalışmanın kapsamlı bir deneysel sonucunu göstermektedir. Aykırı değer noktaları, bir artımlı adım kullanılarak P_{99}'dan P_{95}'e seçilmiş ve P_{91}'e kadar iki adıma yükseltilmiştir. Modelin adı altında kayan-nokta hassasiyeti (FP) kullanan orijinal doğruluk bulunmuştur. Kalın değerler, her bit genişliği deneyinde en yüksek doğruluğu temsil etmiştir. Ek olarak, doğruluktaki gelişmeleri gözlemlemek için önerilen nicemlemenin performansı aykırı değerler ayrımı olmaksızın sunulmuştur. W ve A, sırasıyla ağırlıkların ve özellik haritalarının bit genişliğini temsil etmiştir.

Tablo 6.2. ImageNet veri kümesinde nicemlenmiş AlexNet ve ResNet-18 için ve COCO veri kümesinde nicemlenmiş YOLO için doğruluk oranları

Model	W/A	Bölmeksizin	P_{99}	P_{98}	P_{97}	P_{96}	P_{95}	P_{93}	P_{91}
AlexNet FP:57,2	8/8	56,6	56,6	**56,8**	56,7	56,6	56,6	56,7	56,7
	6/6	55,8	56,2	56,2	**56,7**	56,6	56,6	56,4	56,5
	5/5	54,1	54,4	54,6	54,6	54,7	54,7	**54,8**	**54,8**
	4/4	51,2	53,4	54,2	53,7	54,1	54,4	54,4	**54,6**
ResNet-18 FP: 70,7	8/8	68,1	69,4	69,5	**69,7**	69,6	69,6	69,6	69,5
	6/6	67,8	69,1	68,9	69,0	**69,2**	68,9	68,9	68,9
	5/5	64,3	67,1	66,9	67,1	67,1	67,1	67,9	**68,1**
	4/4	59,9	63,3	63,3	64,0	65,1	65,1	65,0	**65,2**
YOLOv3-tiny FP: 24,5	8/8	23,1	23,9	24,0	**24,2**	23,6	24,0	24,0	24,0
	6/6	22,6	23,9	**24,1**	23,9	24,1	24,1	24,0	23,8
	5/5	20,4	24,1	24,2	23,9	24,1	24,0	**24,2**	24,0
	4/4	20,1	23,0	23,1	23,2	23,2	23,0	23,2	**23,3**

Sonuçlar, ağırlıklar ve özellik haritaları için 8-bit kullanıldığında, önerilen yöntemin kayan-nokta tabanlı modele yakın bir performans elde ettiğini göstermiştir. Ancak ConNN modeli 4-bit olarak nicelleştirildiğinde önerilen yöntemin performansının önemli ölçüde düştüğü görülebilir. Ayrıca büyük model, 4-bit nicemleme kullanıldığında küçük modellerden daha fazla etkilenmiştir. Örneğin ResNet-18, doğruluğu %16 oranında azaltırken, AlexNet üzerinde değerlendirildiğinde doğruluk yalnızca %10 oranında azalmıştır.

Deneysel sonuçlar, aykırı değer ağırlıklarının ayrılması yönteminin logaritmik nicemleme için yararlı olduğunu göstermiştir. Nicemlemenin verimliliği, ağırlık ayrımı olmaksızın nicemleme yönteminin kullanılmasına kıyasla daha da iyileştirilmiştir. Teorik olarak, yüzdelik noktası azaldıkça doğruluk artırılmalıdır. Ancak yüzdelik noktasını azaltarak, nicemlenmiş model her zaman daha yüksek doğruluk sağlamadığı ortaya çıkmıştır. En yüksek doğruluğu sağlayan en uygun yüzdelik noktasını belirlemek için net bir eğilim yoktur. Bununla birlikte, önerilen yöntemin, modeli düşük bit genişliklerinde nicelleştirirken en yüksek doğruluğu elde etmek için daha derin bir yüzdelik noktası gerektirdiği belirtilebilir.

Deneysel sonuçlar, önerilen ağırlık ayrımının, büyük bit genişlikleri ile kullanıldığında doğruluğu geliştirdiğini göstermiştir. Önerilen yöntem, 5-bit nicemleme gerçekleştirirken doğruluğu yaklaşık %1'e kadar iyileştirebilmiştir. Diğer taraftan, önerilen yöntem düşük bit genişlikli nicemleme için kullanışlıdır. Sonuçlar, modelde 4-bit nicemleme kullanıldığında doğrulukta önemli bir gelişme göstermiştir.

6.5.2. Kaynak Yükü Analizi

Büyük miktarda ağırlık ayırmak, dahili belleğin maliyetini artırmıştır. Bu ek gereksinim, önerilen yöntemdeki nöron çoğaltması nedeniyle ortaya çıkmıştır. Bu kaynak yükü, ConNN modellerini FPGA' gibi sınırlı kaynak cihazlarına dağıtmak için bir zorluk sunmuştur. Bu nedenle, önerilen yöntemi FPGA üzerinde uygulamak adına en uygun çözümü bulmak için doğruluk kaynak genel ödünleşimleri incelenmiştir. Bu nedenle, önerilen yöntemin performansını değerlendirmek için bir ölçü olarak bellek verimliliği gösterilmiştir. Bellek verimliliği aşağıdaki gibi ifade edebilir.

$$\text{Bellek verimliliği} = \frac{\text{Doğruluk oranı iyileştirmesi}}{\text{Bellek yükü}} \tag{6.3}$$

Burada doğruluk oranı iyileştirmesi, aykırı değer ayrımı olmadan nicemlenmiş modelin doğruluğuna kıyasla aykırı değer ayrımı kullanan nicemlenmiş modelin doğruluğudur. Bellek yükü, çoğaltma özellik haritaları dahil olmak üzere bellek gereksinimlerinin sayısının orijinal bellek gereksinimlerinin sayısına oranıdır.

Çoğaltılan özelliği haritalarının sayısı sabit olduğu için bellek yükü hesaplanabilir. Örneğin, P_{99} kullanmak, orijinal ağırlık ve özellik harita sayısını %0,01 oranında artırabilmiştir. Bu nedenle, P_{99}'daki aykırı noktayı kullanmak için bellek yükü 1.01'dir. Aynı şekilde, diğer aykırı değerler için bellek yükü hesaplanabilmiştir. Tablo 6.3'te farklı yüzdelik kullanıldığında nicemlenmiş modellerin bellek verimliliği sonuçlarını sunulmuştur. Kalın ve altı çizili puanlar, sırasıyla her bit genişliği için en yüksek bellek verimliliğini ve en yüksek doğruluğu temsil eder.

Tablo 6.3. Aykırı değer ayrıma yöntemini kullanılırken bellek verimliliği

Model	W/A	Bölmeksizin	P_{99}	P_{98}	P_{97}	P_{96}	P_{95}	P_{93}	P_{91}
AlexNet FP:57,2	8/8	**1,00**	0,99	0,98	0,97	0,96	0,95	0,94	0,92
	6/6	**1,00**	**1,00**	0,99	0,99	0,98	0,97	0,94	0,93
	5/5	**1,00**	**1,00**	0,99	0,98	0,97	0,96	0,95	0,93
	4/4	1,00	**1,03**	1,04	1,02	1,02	1,01	0,99	0,98
ResNet-18 FP: 70,7	8/8	1,00	**1,01**	1,00	0,99	0,98	0,97	0,96	0,94
	6/6	1,00	**1,01**	1,00	0,99	0,98	0,98	0,95	0,93
	5/5	1,00	**1,03**	1,02	1,01	1,00	0,99	0,99	0,97
	4/4	1,00	**1,14**	1,13	1,12	1,11	1,10	1,08	1,07
YOLOv3-tiny FP: 24,5	8/8	1,00	**1,02**	**1,02**	**1,02**	1,00	0,99	0,97	0,95
	6/6	1,00	**1,05**	**1,05**	1,03	1,03	1,01	0,99	0,97
	5/5	1,00	**1,17**	1,16	1,14	1,14	1,12	1,11	1,08
	4/4	1,00	**1,13**	**1,13**	1,12	1,11	1,09	1,08	1,06

Deneysel sonuçlara göre, önerilen yöntemin 8 bitlik nicemlemeye uygulanması, yalnızca nicemlemenin kullanılmasından biraz daha yüksek bellek verimliliği sağlamıştır. Bu sonuçlar ayrıca AlexNet ve ResNet-18 için en yüksek doğruluğun elde edilmesinin ek bellek kaynakları gerektirdiğini göstermiştir. Önerilen çalışma, düşük bit nicemleme ile gerçekleştirildiğinde daha iyi sonuçlar vermiştir. 4-bit nicemleme ile değerlendirme yapıldığında, tek başına nicemleme yönteminin kullanılmasına kıyasla önemli ölçüde daha yüksek bellek verimliliği sağlamıştır. Önerilen yöntem ağırlık ayrımı olmaksızın 4-bit nicemleme yöntemiyle karşılaştırıldığında, önerilen yöntem önemli ölçüde daha yüksek bellek verimliliği elde etmiştir.

Deneysel sonuçlar ayrıca aykırı değer sayısının artmasının bellek verimliliğini etkilediğini de göstermiştir. 8-bit genişliğinde nicemleme için P_{91}'in aykırı değer noktası uygulandığında, bellek verimliliği en yüksek puana kıyasla yaklaşık %8 azalmıştır. Diğer

taraftan, 4-bit nicemleme gerçekleştirildiğinde, ResNet-18 ve AlexNet için bellek verimliliği yaklaşık %3 oranında azalmıştır.

6.5.3. Nicemleme Yöntemlerinin Performans Karşılaştırması

Önerilen yöntemin etkinliğini farklı model boyutlarında sunmak için ResNet-18 ve ResNet-50 uygulanmıştır. Önerilen yöntemi kullanan nicelleştirilmiş ConNN modellerinin diğer eğitim sonrası nicemleme yöntemlerine kıyasla ilk 1 doğruluk oranı Tablo 6.4'te gösterilmektedir. FP, modelin orijinal doğruluğunu gösterir. W ve A, sırasıyla ağırlıkların bit genişliğini ve özellik haritalarını belirtir.

Tablo 6.4. ResNet-18 ve ResNet-50 üzerinde değerlendirilen eğitim sonrası nicemleme yöntemlerinin performansının karşılaştırması

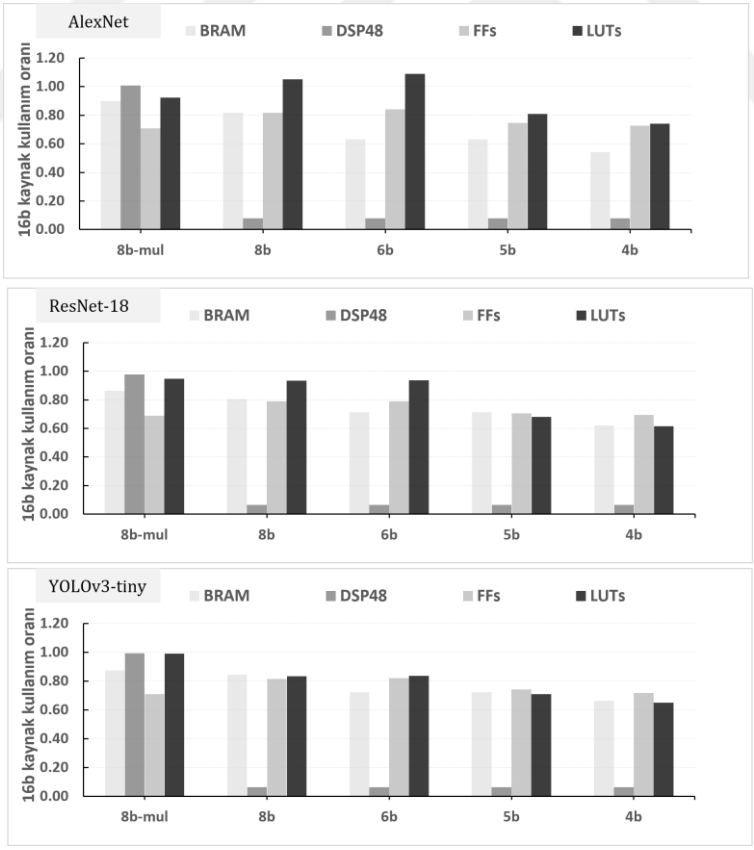
Yöntem	W/A bit	ResNet-18 (FP: 70,7)	ResNet-50 (FP: 75,8)
(Huang ve diğ., 2021)	8/8	67,61	74,06
(R. Zhao ve diğ., 2019)		69,38	75,9
(P. Wang ve diğ., 2020)		69,74	75,96
Önerilen yöntem		69,54	75,1
(R. Zhao ve diğ., 2019)	8/4	66,36	50,39
(Nahshan ve diğ., 2021)		66,3	74,8
Önerilen yöntem		67,32	71,8
(R. Zhao ve diğ., 2019)	4/4	34,85	-
(P. Wang ve diğ., 2020)		67,56	73,1
(Nahshan ve diğ., 2021)		60,3	70
Önerilen yöntem		66,67	60,12

Her iki modelde de değerlendirilen 8-bit kullanan önerilen yöntem, değiştirilmiş bir logaritmik nicemleme öneren (Huang ve diğ., 2021) ile karşılaştırıldığında daha iyi performans vermiştir. Önerilen yönteme benzer şekilde (R. Zhao ve diğ., 2019)'de sunulmuştur. Ancak doğrusal nicemleme için ağırlıkların ayrılmasını önermiştir. Önerilen yöntem, ResNet-18 ve ResNet-50 üzerinde 8-bit nicemleme gerçekleştirirken (R. Zhao ve diğ., 2019)' den daha iyi performans göstermiştir. Önerilen yöntem, 4-bit nicemleme kullandığında (R. Zhao ve diğ., 2019) yönteminden önemli ölçüde daha iyi doğruluk sağlamıştır. Eğitim sonrası nicemleme yöntemi (Nahshan ve diğ., 2021; P. Wang ve diğ., 2020)'de sunulmuştur. Önerilen yöntemi, 8-bit nicemleme kullanıldığında diğer araştırmalarla karşılaştırılabilir bir doğruluk elde etmiştir. Özellik haritalarının bit

genişliğini azaltılması doğruluğu etkilemiştir. Özellik eşlemeleri için bit genişliği 4 bit olduğunda model doğruluğu azalmıştır. Ancak önerilen yöntem, ağırlıklar üzerinde 8-bit nicemleme gerçekleştirirken, önceki çalışmalarla aynı düzeyde performans elde etmiştir. Sonuçlar ayrıca önerilen çalışmanın ağırlıklar ve özellik haritaları üzerinde 4-bit nicemleme kullanıldığında diğer araştırmalarla karşılaştırılabilir doğruluk sağladığını göstermiştir. Ancak önerilen yöntem, ağırlıklar ve özellik haritaları üzerinde 4-bit nicemleme ile gerçekleştirildiğinde ResNet-50'nin doğruluğunu önemli ölçüde azaltmıştır.

6.6. Hızlandırıcının Kaynak Kullanımı

Şekil 6.6 farklı ConNN modellerinde değerlendirilen ayırma aykırı değerler yöntemiyle logaritmik nicemleme gerçekleştirirken önerilen hızlandırıcının kaynak kullanımını göstermektedir. Bu sonuç, 16-bit hızlandırıcı ile karşılaştırıldığında kaynak kullanımını göstermiştir. 8b-mul, çarpma kullanan 8-bit hızlandırıcı belirtir.



Şekil 6.6. Farklı ConNN modellerinde değerlendirildiğinde 16-bit hızlandırıcıya kıyasla kaynak kullanım oranı

Sonuçlar, bit genişliklerinin azaltılmasının hızlandırıcının kaynak kullanım sayısını azaltabileceğini göstermiştir. Qset ile logaritmik nicemlemenin kullanılması LUT kullanımında bir gelişme sağlamıştır. Hızlandırıcı 8, 6, 5 veya 4-bit ile uygulandığında, 16-bit hızlandırıcılara kıyasla LUT sayısı azalmıştır. Daha önce tasarlanan hızlandırıcının 16 bit hızlandırıcıdan daha fazla LUT kaynağı tüketmesi nedeniyle, önerilen bu yöntem önceki nicemleme yönteminden daha üstündür. Özetle, bit kesri kullanılarak Qset ile logaritmik nicemleme daha az LUT gerektirmiştir çünkü arama tablosu tamsayı değerlerini depolamak için gerekli değildir ve çarpma hesaplaması için arama tablosu daha az girdi gerektirmiştir.

6.7. Sonuçlar

Tezde düşük bit genişlikleri uygulanırken yüksek çözünürlüklü logaritmik nicemleme yönteminin performansını geliştirmek için bir aykırı ağırlık ayırma tekniği önerilmiştir. Önerilen yöntem, ağırlık ayırma tekniğini kullanarak ağırlık dağılımını yeniden şekillendirerek logaritmik nicemleme için uygun bir girdi sağlamıştır. Deneysel sonuçlar, önerilen yöntemin 4-bit ile nicemleme yapılırken logaritmik nicemlemenin performansını önemli ölçüde iyileştirdiğini kanıtlamıştır. Bellek verimliliği sonuçları, bu yöntemin bellek yükü olmadan kullanılabileceğini göstermiştir. Ayrıca önerilen yöntem, çarpma tabanlı 16-bit hızlandırıcıdan daha fazla kaynak verimliliği sağlamıştır.

7. NESNE TANIMA UYGULAMALARI

Bu bölümde, ConNN gerçekleştirmek için FPGA kullanımı gösterilmiştir. FPGA üzerinde donanım hızlandırıcı oluşturmak için geliştirme araçları daha önceki bölümlerde tanıtılmıştır. Bu bölümde de görüntü sınıflandırma ve nesne algılamanın deneysel sonuçları görselleştirme yoluyla sunulmuştur.

7.1. Görüntü Sınıflandırma

Görüntü sınıflandırmanın amacı, görüntüdeki nesneleri tanımlamaktır. Bu, bilgisayar görüsünde klasik bir hesaplama görevidir. Daha önce bu görevi gerçekleştirmek için makine öğrenimi yöntemleri kullanılmıştır. Son zamanlarda ConNN algoritmaları, klasik makine öğrenme yöntemlerinden daha iyi bir performans vermiş ve görüntü sınıflandırmasında önemli bir gelişme göstermiştir. Bu tezde, görüntü sınıflandırması için AlexNet ve ResNet modelleri uygulanmıştır. AlexNet modeli, 5 konvolüsyonel katmanı ve 3 FC katmanından oluşan küçük bir ConNN modelini temsil edecek şekilde seçilmiştir. ResNet modeli ise 18'den fazla katmandan oluşan karmaşık bir ConNN modeline örnek olarak seçilmiştir.

AlexNet ve ResNet-18'in çıktı katmanı, Softmax aktivasyon fonksiyonu tarafından normalize edilen 1000 çıktı özelliği haritalarına sahiptir. Softmax, görüntü sınıflandırma modelinin çıktı katmanına uygulanan bir tür aktivasyon fonksiyonu olarak kabul edilmiştir. Softmax fonksiyonu, her nesne sınıfının olasılıklarını içeren çıktı vektörü sağlamıştır. Modelin performansını değerlendirmek için, gerçek sınıfa kıyasla tahmin edilen sonuçların doğruluğuna dayalı olarak en iyi bir ve en iyi 5 doğruluk oranı hesaplanmıştır. En iyi bir, gerçek sınıfla karşılaştırıldığında en yüksek tahminin doğruluğudur. En iyi 5 ise, gerçek sınıfla karşılaştırıldığında ilk 5 tahminin doğruluğudur.

7.2. Nesne Tanıma ve Algılama

Nesne algılama, bir görüntüden insan, hayvan veya araç gibi belirli bir nesne türünün algılanması üzerinde çalışan bilgisayarlı görü görevlerinden biridir. Nesne algılamanın amacı, görüntüdeki nesnelerin nerede ve ne tür olduğunu anlayabilmektir. Nesne algılamanın temel sorunu sınıflandırma ve konumlandırma görevleridir. Nesne algılamadaki zorluklar sadece bu görevleri çözmek için değildir, aynı zamanda algılama

hızı, nesne döndürme veya nesne ölçeği sorunları da nesne algılamada zorluklar yaratmaktadır. Bu tezde, nesne algılama için ConNN modelinin bir örneği olarak YOLOv3 tiny modeli uygulanmıştır.

Bu model 80 nesne sınıfını algılamak için Microsoft'un COCO veri kümesi üzerinde eğitilmiştir. YOLOv3-tiny, nesnenin konumunu ve sınıflandırma olasılığını içeren bir tahmin sonucu sunmuştur. Son katmanın çıktı özellik haritaları, görüntü boyutuna göre sınırlayıcı kutunun tahmini yüksekliği (h) ve genişliği (w), bir nesnenin merkez koordinatı (x, y), her bir sınır kutusunun güven değeri ve bir nesneye sahip olma olasılıklarından oluşmuştur. Sınırlayıcı kutunun genişliği (w) ve yüksekliği (h) gerçek görüntü boyutuna göre hesaplanmıştır. YOLOv3-tiny'ün çıktıları sınıf sayısı göre hesaplanabilir. Örneğin, model 80 sınıf öngördüyse, çıktı sayısı $1\times1(4+1+80)$ olacaktır.

7.3. ConNN Çerçevesi

ConNN modelleri, C++ ve CUDA programlama ile yazılmış açık kaynaklı bir sinir ağları çerçevesi olan Darknet adlı bir çerçeve üzerinde uygulanmıştır. Bu çerçeve, evrişim katmanı, FC katmanı, Havuzlama katmanı ve ayrıca sinir ağları algoritmalarıyla ilgili diğer hesaplama yöntemlerini destekler. Bu çerçeve ayrıca katman özelleştirmesi için esnek parametre yapılandırmaları sağlar. Yapılandırmada birkaç etkinleştirme yöntemi mevcuttur. Ayrıca Darknet, eğitim aşamasını hızlandırmak için GPU ile birlikte kullanılabilir. Tezde uygulanan tüm modeller bu çerçevede eğitilmiştir. Çerçeve, yalnızca C++ derleyicisi gerektirdiğinden çeşitli platformlarla uyumludur. Bu çerçeveyi sınırlı kaynak donanım cihazlarında uygulamak uygundur. Değerlendirmede FPGA tarafından gerçekleştirilen yalnızca ConNN çıkarımıdır. Bu nedenle, eğitim aşaması ile ilgili işlevler ve değişkenler kaldırılmıştır.

7.4. FPGA Uygulamaları Geliştirme Programları

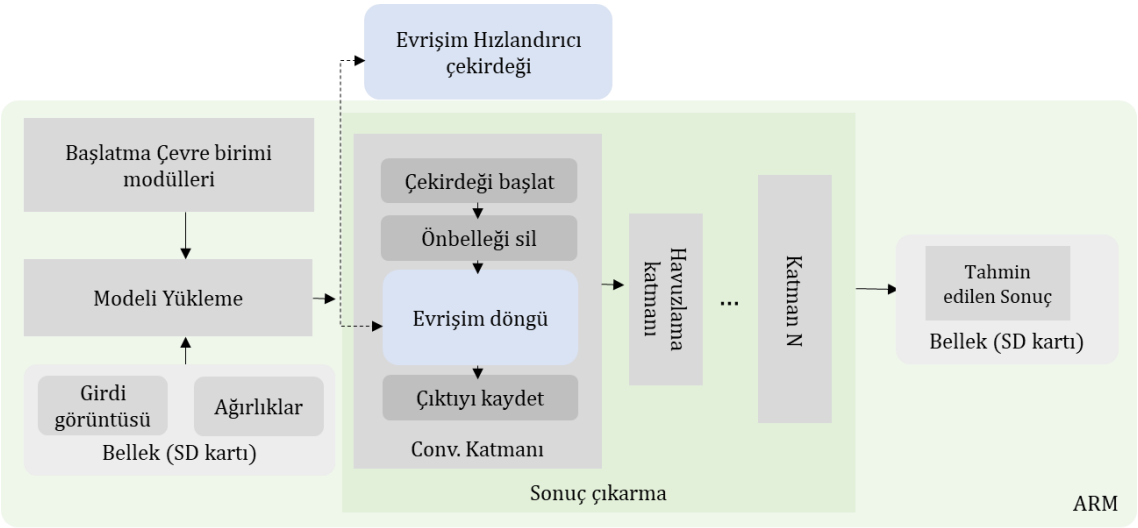
Bu tezde, Xilinx satıcısından XC7Z020 FPGA tabanlı bir geliştirme cihazı olan Zedboard kullanılmıştır. Xilinx, bu kart üzerinde sayısal bir devre tasarlamak için geliştirici araçları sağlar. Bu tezde önerilen hızlandırıcının uygulanması için Vivado HLS, Vivado ve Xilinx SDK araç olarak kullanılmıştır. Vivado HLS, C veya C++ programlama dilini HDL veya

RTL'ye sentezlemek için kullanılan bir araçtır. Bu araç, hızlandırıcı çekirdeğini ve RTL tasarımını oluşturmak için kullanılır.

Vivado, FPGA'da donanım tasarımlarını uygulamak için ana yazılımdır. Vivado, donanım modüllerini hedeflenen cihazlarla eşleştirerek HDL sentezleyebilir ve ayrıca tasarımı FPGA'ya programlayabilir. Vivado, işlemci, hızlandırıcı ve PS ile PL bölümleri arasındaki bağlantıdan oluşan entegre tasarımı oluşturmak için kullanılır. Bu araç aynı zamanda işlemci ve arabirim bağlantı noktalarının saat hızını yapılandırmak için de kullanılır. Xilinx SDK, gömülü cihazlar için C veya C++ uygulamaları oluşturmak için Xilinx tarafından üretilen bir yazılım geliştirme kitidir. Bu araç, ARM işlemcisi için bir program oluşturmaya yönelik yazılımdır. Tezde, işlemci üzerinde yazılım geliştirmek ve hızlandırıcıyı kontrol etmek için Xilinx SDK kullanılmıştır.

7.5. FPGA'da İş Akışı Süreci

ConNN hesaplaması için sistemin çalışma zamanı iş akışı Şekil 7.1'de sunulmuştur. Güç verildikten sonra Zynq işlemci PL bölümünde hızlandırıcı çekirdeği de dahil olmak üzere gerekli çevresel modülleri başlatır. Daha sonra işlemci, ConNN modelinin yapısına göre bir katman oluşturur. Evrişim katmanı için işlemci, cihaza takılan SD karttan ağırlık değerini okur ve ardından DDR belleğe yazar.



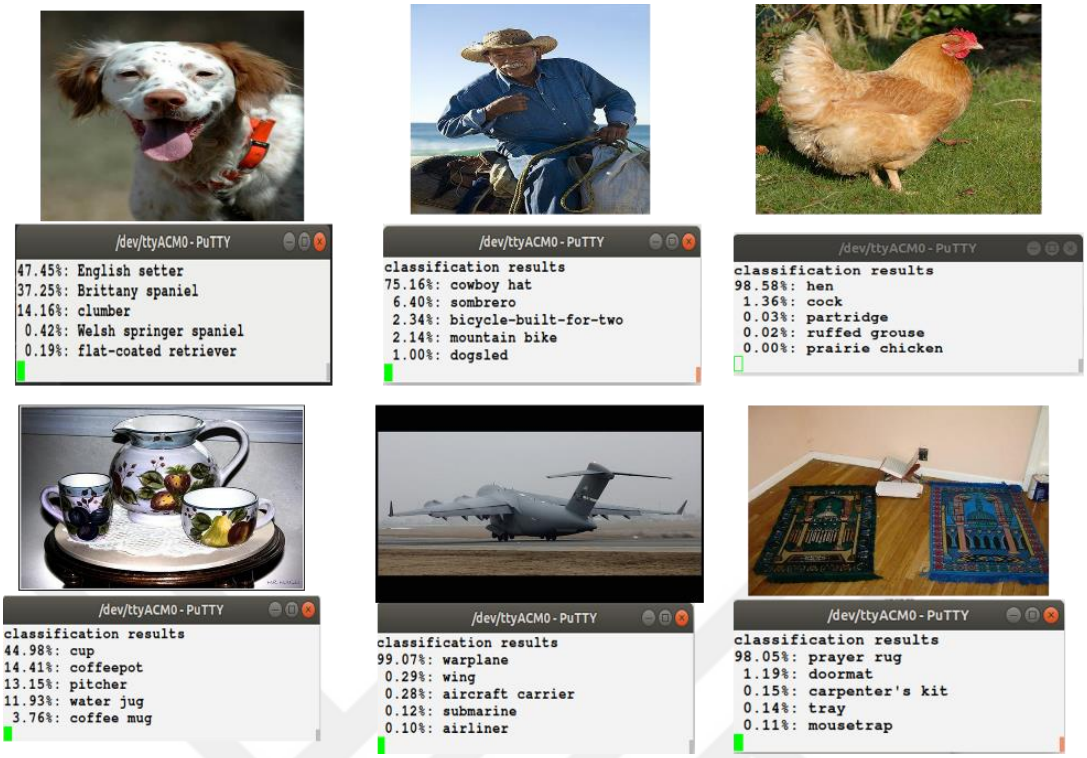
Şekil 7.1. Hızlandırıcının iş akış çalışma zamanı

Tüm katmanlar oluşturulduğunda işlemci ConNN çıkarımını gerçekleştirmeye başlar. Evrişim katmanının yürütülmesi gerekiyorsa, işlemci çekirdeği başlatmak için hızlandırıcı çekirdeğin durumunu kontrol eder. Hızlandırıcıya veri göndermeden önce işlemci önbelleği temizler. Daha sonra konvolüsyonel gerçekleştirmek için PL bölümündeki hızlandırıcı çekirdeğe girdi özellik haritaları ve ağırlıklar gönderilir. Sonuçlar işlemciye geri gönderilir ve DDR belleğe aktarılır. Havuzlama katmanının yürütülmesi gerekiyorsa, işlemci bu katmanı kendi başına çalıştıracaktır. İş akışı çalışma zamanı son katmana kadar devam eder. Sonuç belleğe gönderilir ve işlemci bu sonucu UART terminali aracılığıyla kullanıcıya gösterir. Bu tahmin sonucu, bilgisayarda etiketli bir görüntü olarak görüntülenmek üzere SD karta da kaydedilir.

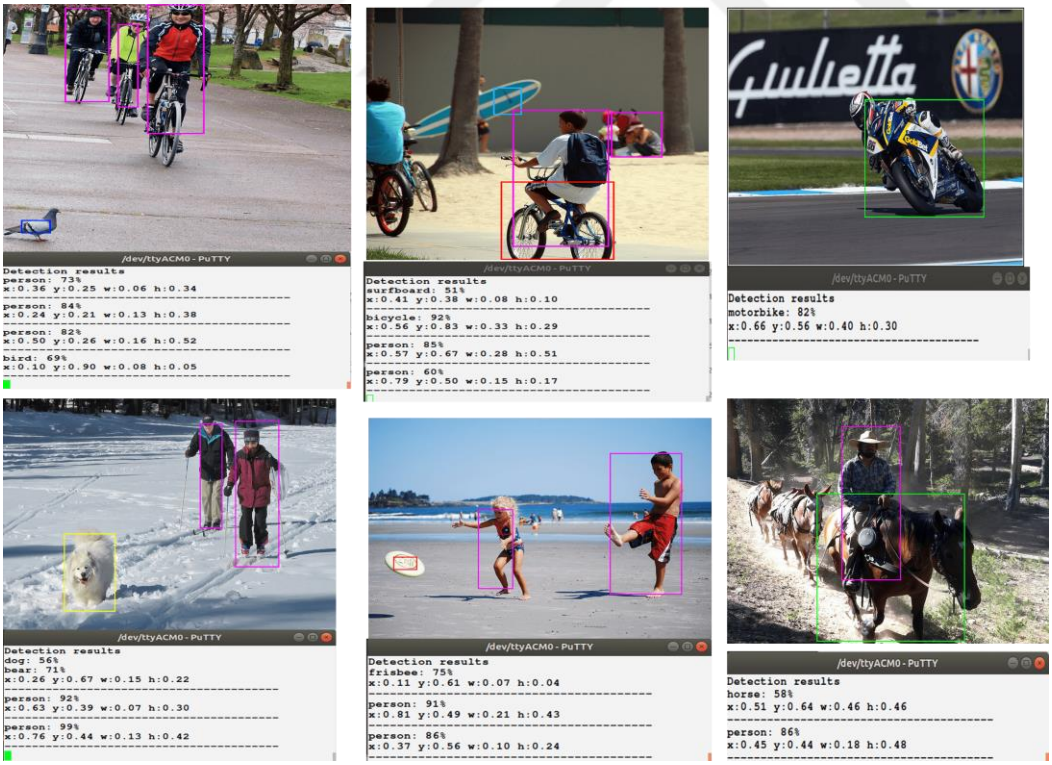
7.6. Deneysel Sonuçlar

Bilgisayar, seri port üzerinden deneysel cihaza bağlanmıştır. Girdi görüntüsü ve ağırlık verilerini depolamak için SD kart kullanılmıştır. Bu dosyaları, kart üzerindeki işlemciye bağlı olan SD kartta tutulmuştur. Yürütme tamamlandıktan sonra işlemci sonuçları SD karta geri yazmıştır. Bu nedenle deneysel cihaz, yalnızca etiketlenmiş sonucu seri terminal aracılığıyla sunabilir. Darknet çerçevesi kullanılarak, SD kartta depolanan çıktı dosyası, etiketli bir görüntü olarak görüntülenmesi için bilgisayarda okunmuştur. Çıktı dosyası, son katmanın çıktı özellik haritalarından oluşmuştur. Tahmin edilen çıktıyı bir görüntü olarak görüntülemek için Darknet çerçevesi ve veri düzenleme işlevi kullanılmıştır. YOLOv3-tiny, COCO veri kümesinde Darknet kullanılarak eğitilmiştir. Darknet çerçevesi, ResNet-18 modelini ImageNet veri kümesi üzerinde eğitmek için de kullanılmıştır. Bu tezde ConNN modelleri yazar tarafından eğitilmemiştir. Darknet çerçeve web sitesinde önceden eğitilmiş YOLOv3-tiny modeli bulunabilir.

Şekil 7.2, görüntü sınıflandırması için ResNet-18 kullanıldığında elde edilen sonuçların bir örneği göstermiştir. YOLOv3-tiny modelinin algılama sonuçları Şekil 7.3'te sunulmuştur. Görüntünün altında, nesnenin koordinatları (x,y) ve sınırlayıcı kutunun boyutları (w,h) dahil olmak üzere sonuçları bulunmuştur.



Şekil 7.2. ResNet-18 görüntü sınıflandırma sonuçları örneği



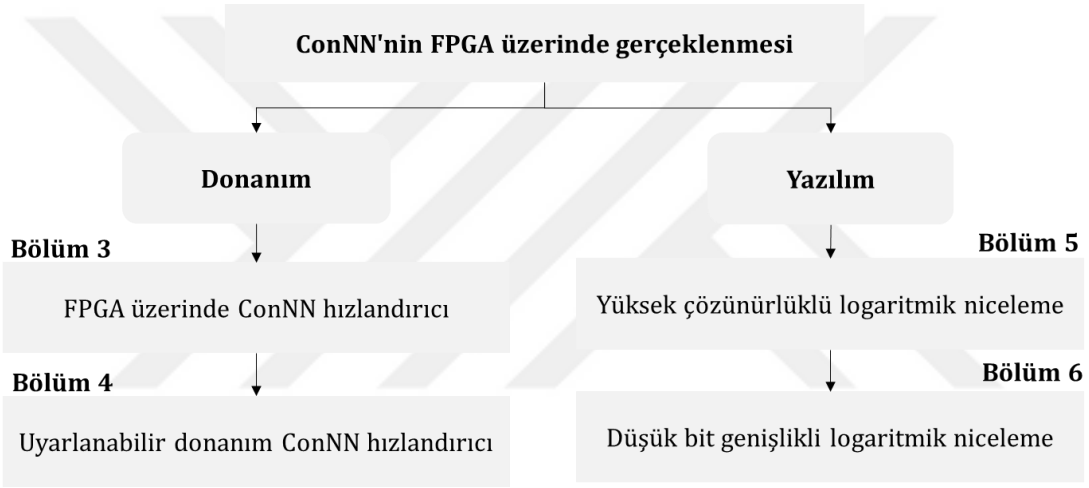
Şekil 7.3. YOLOv3-tiny algılama sonuçları örneği

7.7. Sonuçlar

Bu bölümde, görüntü sınıflandırma ve nesne tespiti amacıyla FPGA üzerinde ConNN uygulanmıştır. ConNN modelinin uygulanması için çerçeve açıklanmıştır. Ek olarak, bir FPGA üzerinde donanım tasarımları oluşturmak için geliştirme araçları sunulmuştur. Son kısımda deneysel sonuçların görselleştirilmesi anlatılmıştır.

8. SONUÇLAR VE ÖNERİLER

Gömülü donanımda Evrişimsel Sinir Ağlarının (ConNN) hızlandırılması, akademik ve gerçek dünya uygulamalarda ilgi görmüştür. FPGA, güç verimliliğindeki iyileştirmeleri ve kullanıcı dostu geliştirme araçları nedeniyle ConNN uygulamaları için kullanılan güçlü gömülü platformlardan biridir. Bu tezde, kaynak yönünden kısıtlı küçük ölçekli FPGA cihazlarında ConNN modelini uygulamak için hem yazılım hem de donanım bölümlerinde verimli teknikler önerilmiştir. Ayrıca görüntü sınıflandırma ve nesne algılama uygulamalarını desteklemek için FPGA üzerinde uygulanabilecek pratik ve basit stratejiler de sunulmuştur. Tezin özeti Şekil 8.1'de gösterilmiştir.



Şekil 8.1. Tezin özeti

Bölüm 3, FPGA tabanlı aygıtta evrişimi hızlandırmak için etkili teknikler göstermiştir. Evrişim hızlanmasını analiz etmek için döngü optimizasyon tekniklerinin sayısal karakterizasyonu da sağlanmıştır. Bunun yanı sıra, ConNN hızlandırıcı için bir donanım yapısı tasarımı önerilmiştir. Önerilen yöntemin etkinliğini göstermek için DSP ve LUT verimliliğini içeren kaynak verimliliği faktörleri kullanılmıştır. Önerilen hızlandırıcı, diğer araştırmalarla karşılaştırılabilir kaynak verimliliği elde etmiştir.

4. bölümde, ConNN modellerini gerçekleştirmek için esnek bir donanım yapısı sağlamak üzere uyarlanabilir bir donanım ConNN hızlandırıcısı önerilmiştir. Uyarlanabilir hızlandırıcı, yapısı hesaplama modüllerinin sayısı ve katmandaki veri sayısı gibi katman özelliklerine göre yeniden yapılandırılmıştır. Çalışma zamanı sırasında hızlandırıcının donanım yapısını yeniden yapılandırmak için kısmi yeniden yapılandırma tekniği

uygulanmıştır. Önerilen uyarlamalı hızlandırıcı, statik tasarıma kıyasla bellek verimliliğinde bir gelişme sağlamıştır.

Donanım açısından verimli teknikler sunulduktan sonra, ConNN hızlandırması için yazılım tabanlı teknikler de önerilmiştir. Bölüm 5, ConNN hesaplamalarını basitleştirmek için yüksek çözünürlüklü bir logaritmik nicemleme sunmuştur. Önerilen nicemleme, çok düşük bit genişlikli bir nicemleme kullanarak ConNN modelini uygularken tatmin edici doğruluk elde etmek için yüksek çözünürlüklü bir logaritmik ölçek kullanmıştır. Önerilen nicemleme ayrıca, nicemlenmiş değerler arasındaki çarpma işleminin bir bit kaydırma işlemi kullanılarak değiştirilebilmesi nedeniyle donanım dostu bir işlem getirmiştir. Bu nedenle, bit kaydırma tabanlı bir ConNN hızlandırıcı önerilmiştir. Sonuçlar, DSP verimliliğinin önemli ölçüde iyileştiğini göstermiştir. Ancak, önerilen hızlandırıcı daha fazla sayıda LUT kaynağı gerektirmiştir. Dolayısıyla, LUT verimliliği, çarpma tabanlı hızlandırıcıya kıyasla düşmüştür.

Bölüm 6, ağırlık dağılımını yeniden şekillendirerek logaritmik nicemlemenin performansını artırmaya yönelik bir yöntem olan aykırı değerlerin farkında bir nicemleme yöntemi sunmuştur. Bu yöntem, nicemlenmiş ConNN modellerinin doğruluğunu artırabilen basit ama etkili bir yöntemdir. Önerilen yöntem, ağırlık ayırma tekniğini kullanarak ağırlığı iki alt değere bölmüştür. Girdi değeri düşük olduğundan logaritmik nicemlemenin performansı iyileştirilmiştir. Bunun yanında, bit kaydırma tabanlı bir ConNN hızlandırıcı önerilmiştir. Önerilen hızlandırıcı, çarpma tabanlı hızlandırıcıya kıyasla DSP ve LUT verimliliğinde iyileştirmeler sağlamıştır.

Gelecekte, nicemleme yönteminin sınırlarını genişletmek için bir optimal bit genişlikli nicemleme araştırması yapılmalıdır. Uyarlanabilir bir donanım hızlandırıcısını optimal bit genişliği nicemleme ile birleştirerek kaynakların verimliliğini artırmak mümkündür.

KAYNAKLAR

Abdel-Hamid, O., Mohamed, A., Jiang, H., Deng, L., Penn, G., Yu, D. (2014). Convolutional Neural Networks for Speech Recognition, *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, *22*(10), 1533-1545. DOI:10.1109/TASLP.2014.2339736

Amodei, D., Ananthanarayanan, S., Anubhai, R., Bai, J., Battenberg, E., Case, C., Casper, J., Catanzaro, B., Cheng, Q., Chen, G. (2016). Deep speech 2: End-to-end speech recognition in english and mandarin, *International conference on machine learning*, New York, ABD, 19-24 Haziran 2016.

Aydonat, U., O'Connell, S., Capalija, D., Ling, A. C., Chiu, G. R. (2017). An opencl™ deep learning accelerator on arria 10, *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, ABD, 22 - 24 Şubat 2017.

Banner, R., Nahshan, Y., Hoffer, E., Soudry, D. (2018). ACIQ: Analytical clipping for integer quantization of neural networks, *arXiv preprint., arXiv:1810.05723*.

Banner, R., Nahshan, Y., Soudry, D. (2019). Post training 4-bit quantization of convolutional networks for rapid-deployment, *Advances in Neural Information Processing Systems*, ABD, 8-14 Aralık 2019.

Boutros, A., Yazdanshenas, S., Betz, V. (2018). You cannot improve what you do not measure: FPGA vs. ASIC efficiency gaps for convolutional neural network inference, *ACM Transactions on Reconfigurable Technology and Systems (TRETS)*, *11*(3), 1-23. DOI:10.1145/3242898

Cai, J., Takemoto, M., Nakajo, H. (2018). A deep look into logarithmic quantization of model parameters in neural networks, *Proceedings of the 10th International Conference on Advances in Information Technology*, Bangkok, Tayland, 10-13 Aralık 2018.

Cai, Y., Yao, Z., Dong, Z., Gholami, A., Mahoney, M. W., Keutzer, K. (2020). Zeroq: A novel zero shot quantization framework, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Seattle, ABD, 13-19 Haziran 2020.

Chang, S. E., Li, Y., Sun, M., Shi, R., So, H. K., Qian, X., Wang, Y., Lin, X. (2021). Mix and match: A Novel Fpga-centric Deep Neural Network Quantization Framework, *2021 IEEE International Symposium on High-Performance Computer Architecture (HPCA)*, Seoul, South Korea, 27 Şubat-3 Mart 2021.

Chen, Y., He, J., Zhang, X., Hao, C., Chen, D. (2019). Cloud-DNN: An Open Framework for Mapping DNN Models to Cloud FPGAs, *Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, New York, ABD, 24-26 Şubat 2019.

Courbariaux, M., Hubara, I., Soudry, D., El Yaniv, R., Bengio, Y. (2016). Binarized neural networks: Training deep neural networks with weights and activations constrained to+ 1 or-1, *arXiv preprint arXiv:1602.02830*.

Farhadi, M., Ghasemi, M., Yang, Y. (2019). A novel design of adaptive and hierarchical convolutional neural networks using partial reconfiguration on fpga, *2019 IEEE High Performance Extreme Computing Conference (HPEC)*, Waltham, ABD, 24 - 26 Eylül 2019.

Feng, G., Hu, Z., Chen, S., Wu, F. (2016). Energy-efficient and high-throughput FPGA-based accelerator for Convolutional Neural Networks, *2016 13th IEEE International Conference on Solid-State and Integrated Circuit Technology (ICSICT)*, Hangzhou, China, 25-28 Ekim 2016.

Fowers, J., Ovtcharov, K., Papamichael, M., Massengill, T., Liu, M., Lo, D., Alkalay, S., Haselman, M., Adams, L., Ghandi, M., Heil, S., Patel, P., Sapek, A., Weisz, G., Woods, L., Lanka, S., Reinhardt, S. K., Caulfield, A. M., Chung, E. S., Burger, D. (2018). A Configurable Cloud-Scale DNN Processor for Real-Time AI, *2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA)*, Los Angeles California, ABD, 2-6 Haziran 2018.

Gong, L., Wang, C., Li, X., Zhou, X. (2020). Improving HW/SW Adaptability for Accelerating CNNs on FPGAs Through A Dynamic/Static Co-Reconfiguration Approach, *IEEE Transactions on Parallel and Distributed Systems*, *32*(7), 1854-1865. DOI:10.1109/TPDS.2020.3046762

Guan, Y., Liang, H., Xu, N., Wang, W., Shi, S., Chen, X., Sun, G., Zhang, W., Cong, J. (2017). FP-DNN: An automated framework for mapping deep neural networks onto FPGAs with RTL-HLS hybrid templates, *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, ABD, 30 Nisan-2 Mayıs 2017.

Guo, K., Sui, L., Qiu, J., Yu, J., Wang, J., Yao, S., Han, S., Wang, Y., Yang, H. (2017). Angel-eye: A Complete Design Flow for Mapping CNN Onto Embedded FPGA, *IEEE transactions on computer-aided design of integrated circuits and systems*, *37*(1), 35-47. DOI:10.1109/TCAD.2017.2705069

He, K., Zhang, X., Ren, S., Sun, J. (2016). Deep residual learning for image recognition, *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas, ABD, 27-30 Haziran 2016.

Huang, C., Liu, P., Fang, L. (2021). MXQN: mixed quantization for reducing bit-width of weights and activations in deep convolutional neural networks, *Applied Intelligence.*, *51*, 4561-4574. DOI:10.1007/s10489-020-02109-0

Jain, A. K. (2010). Data clustering: 50 years beyond K-means, *Pattern Recognition Letters*, *31*(8), 651-666. DOI:10.1016/j.patrec.2009.09.011

Jiang, H., Learned-Miller, E. (2017). Face detection with the faster R-CNN, *2017 12th IEEE international conference on automatic face & gesture recognition (FG 2017)*, Washington DC, ABD, 30 Mayıs-3 Haziran 2017.

Kästner, F., Janßen, B., Kautz, F., Hübner, M., Corradi, G. (2018). Hardware/software codesign for convolutional neural networks exploiting dynamic partial reconfiguration on PYNQ, *2018 IEEE International Parallel and Distributed Processing Symposium Workshops (IPDPSW)*, Vancouver, Kanada, 21-25 Mayıs 2018.

Kim, H., Nam, H., Jung, W., Lee, J. (2017). Performance analysis of CNN frameworks for GPUs, *2017 IEEE International Symposium on Performance Analysis of Systems and Software (ISPASS)*, Santa Rosa, ABD, 24-25 Nisan 2017.

Krizhevsky, A., Sutskever, I., Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks, *Advances in neural information processing systems*, California, ABD, 3-8 Aralık 2012.

Lavin, A., Gray, S. (2016). Fast algorithms for convolutional neural networks, *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas, ABD, 27-30 Haziran 2016.

Lecun, Y., Bottou, L., Bengio, Y., Haffner, P. (1998). Gradient-based learning applied to document recognition, *Proceedings of the IEEE*, *86*(11), 2278-2324. DOI:10.1109/5.726791

Li, H., Lin, Z., Shen, X., Brandt, J., Hua, G. (2015). A convolutional neural network cascade for face detection, *Proceedings of the IEEE conference on computer vision and pattern recognition*, Boston, ABD, 7-12 Haziran 2015.

Li, Y., Dong, X., Wang, W. (2019). Additive powers-of-two quantization: An efficient non-uniform discretization for neural networks, *arXiv preprint., arXiv:1909.13144*.

Liu, W., Anguelov, D., Erhan, D., Szegedy, C., Reed, S., Fu, C. Y., Berg, A. C. (2016). Ssd: Single shot multibox detector, *European conference on computer vision*, Honolulu, Hawaii, 22-25 Temmuz 2016.

Liu, Z., Dou, Y., Jiang, J., Xu, J., Li, S., Zhou, Y., Xu, Y. (2017). Throughput-Optimized FPGA Accelerator for Deep Convolutional Neural Networks, *ACM Transactions on Reconfigurable Technology and Systems*, *10*(3). DOI:10.1145/3079758

Lo, C. Y., Sham, C. W. (2020). Energy efficient fixed-point inference system of convolutional neural network, *2020 IEEE 63rd International Midwest Symposium on Circuits and Systems (MWSCAS)*, Springfield MA, ABD, 9-12 Ağustos 2020.

Lu, L., Liang, Y., Xiao, Q., Yan, S. (2017). Evaluating fast algorithms for convolutional neural networks on FPGAs, *2017 IEEE 25th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Napa, ABD, 30 Nisan-2 Mayıs 2017.

Lu, L., Xie, J., Huang, R., Zhang, J., Lin, W., Liang, Y. (2019). An Efficient Hardware Accelerator for Sparse Convolutional Neural Networks on FPGAs, *IEEE 27th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, San Diego CA, USA, 28 Nisan-1 Mayıs 2019.

Lu, T. Y., Chin, H. H., Wu, H. I., Tsay, R. S. (2020). A Very Compact Embedded CNN Processor Design Based on Logarithmic Computing, *arXiv preprint, arXiv:2010.11686*.

Luo, C., Sit, M.-K., Fan, H., Liu, S., Luk, W., Guo, C. (2020). Towards efficient deep neural network training by FPGA-based batch-level parallelism, *Journal of Semiconductors, 41*(2), 022403. DOI:10.1109/FCCM.2019.00016

Ma, Y., Suda, N., Cao, Y., Vrudhula, S., Seo, J. (2018). ALAMO: FPGA Acceleration of Deep Learning Algorithms with a Modularized RTL Compiler, *Integration, 62*, 14-23. DOI:10.1016/j.vlsi.2017.12.009

Madadum, H., Becerikli, Y. (2022). A Resource-Efficient Convolutional Neural Network Accelerator Using Fine-Grained Logarithmic Quantization, *Intelligent Automation & Soft Computing, 33*(2), 681-695. DOI:10.32604/iasc.2022.023831

Maitra, D. S., Bhattacharya, U., Parui, S. K. (2015). CNN based common approach to handwritten character recognition of multiple scripts, *13th International Conference on Document Analysis and Recognition (ICDAR)*, Tunis, Tunisia, 23-26 Ağustos 2015.

Miyashita, D., Lee, E. H., Murmann, B. (2016). Convolutional neural networks using logarithmic data representation, *arXiv preprint arXiv:1603.01025*.

Moolchandani, D., Kumar, A., Sarangi, S. R. (2021). Accelerating CNN inference on ASICs: A survey, *Journal of Systems Architecture, 113*, 101887. DOI:10.1016/j.sysarc.2020.101887

Nahshan, Y., Chmiel, B., Baskin, C., Zheltonozhskii, E., Banner, R., Bronstein, A. M., Mendelson, A. (2021). Loss aware post-training quantization, *Machine Learning., 110*(11), 1-18. DOI:10.1007/s10994-021-06053-z

Nurvitadhi, E., Sim, J., Sheffield, D., Mishra, A., Krishnan, S., Marr, D. (2016). Accelerating recurrent neural networks in analytics servers: Comparison of FPGA, CPU, GPU, and ASIC, *26th International Conference on Field Programmable Logic and Applications (FPL)*, Lausanne, Switzerland, 29 Ağustos - 2 Eylül 2016.

Nurvitadhi, E., Venkatesh, G., Sim, J., Marr, D., Huang, R., Ong Gee Hock, J., Liew, Y. T., Srivatsan, K., Moss, D., Subhaschandra, S. (2017). Can FPGAs beat GPUs in accelerating next-generation deep neural networks?, *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, Monterey California, ABD, 22-24 Şubat 2017.

Oh, S., Sim, H., Lee, S., Lee, J. (2021). Automated log-scale quantization for low-cost deep neural networks, *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, Nashville TN, ABD, 20-25 Haziran 2021.

Park, E., Yoo, S., Vajda, P. (2018). Value-aware quantization for training and inference of neural networks, *Proceedings of the European Conference on Computer Vision (ECCV)*, Münih, Almanya, 8-14 Eylül 2018.

Putic, M., Venkataramani, S., Eldridge, S., Buyuktosunoglu, A., Bose, P., Stan, M. (2018). Dyhard-dnn: Even more dnn acceleration with dynamic hardware reconfiguration, *Proceedings of the 55th Annual Design Automation Conference*, CA, USA, 24-28 Haziran 2018.

Qasaimeh, M., Denolf, K., Lo, J., Vissers, K., Zambreno, J., Jones, P. H. (2019). Comparing energy efficiency of CPU, GPU and FPGA implementations for vision kernels, *IEEE international conference on embedded software and systems (ICESS)*, Las Vegas NV, ABD, 2-3 Haziran 2019.

Qin, J., Pan, W., Xiang, X., Tan, Y., Hou, G. (2020). A biological image classification method based on improved CNN, *Ecological Informatics*, *58*, 101093. DOI:10.1016/j.ecoinf.2020.101093

Qiu, J., Wang, J., Yao, S., Guo, K., Li, B., Zhou, E., Yu, J., Tang, T., Xu, N., Song, S. (2016). Going deeper with embedded fpga platform for convolutional neural network, *Proceedings of the 2016 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey CA, ABD, 21-23 Şubat 2016.

Rahman, A., Lee, J., Choi, K. (2016). Efficient FPGA Acceleration of Convolutional Neural Networks Using Logical-3D Compute Array, *Design, Automation Test in Europe Conference Exhibition (DATE)*, Dresden, Almanya, 14-18 Mart 2016.

Redmon, J., Divvala, S., Girshick, R., Farhadi, A. (2016). You only look once: Unified, real-time object detection, *Proceedings of the IEEE conference on computer vision and pattern recognition*, Las Vegas NV, ABD,27-30 Haziran 2016.

Redmon, J., Farhadi, A. (2018). Yolov3: An incremental improvement, *arXiv preprint arXiv:1804.02767*.

Russakovsky, O., Deng, J., Su, H., Krause, J., Satheesh, S., Ma, S., Huang, Z., Karpathy, A., Khosla, A., Bernstein, M. (2015). Imagenet large scale visual recognition challenge, *International journal of computer vision*, *115*(3), 211-252. DOI:10.1007/s11263-015-0816-y

Simonyan, K., Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition, *arXiv preprint arXiv:1409.1556*.

Siu, K., Stuart, D. M., Mahmoud, M., Moshovos, A. (2018). Memory Requirements for Convolutional Neural Network Hardware Accelerators, *IEEE International Symposium on Workload Characterization (IISWC)*, Raleigh NC, ABD, 30 Eylül-2 Ekim 2018.

Skrimponis, P., Pissadakis, E., Alachiotis, N., Pnevmatikatos, D. (2020). Accelerating binarized convolutional neural networks with dynamic partial reconfiguration on disaggregated FPGAs, *Parallel Computing: Technology Trends*, *1*(36), 691-700. DOI:10.3233/APC200099

Szegedy, C., Ioffe, S., Vanhoucke, V., Alemi, A. A. (2017). Inception-v4, inception-resnet and the impact of residual connections on learning, *Thirty-first AAAI conference on artificial intelligence*, San Francisco, ABD,4-9 Şubat 2017.

Tan, M., Le, Q. (2019). Efficientnet: Rethinking model scaling for convolutional neural networks, *International conference on machine learning*, Kaliforniya, ABD, 9-15 Haziran 2019.

Touvron, H., Vedaldi, A., Douze, M., Jégou, H. (2020). Fixing the train-test resolution discrepancy: Fixefficientnet, *arXiv preprint arXiv:2003.08237*.

Umuroglu, Y., Fraser, N. J., Gambardella, G., Blott, M., Leong, P., Jahre, M., Vissers, K. (2017). Finn: A framework for fast, scalable binarized neural network inference, *Proceedings of the 2017 ACM/SIGDA international symposium on field-programmable gate arrays*, Monterey California, ABD, 22-24 Şubat 2017.

Vasudevan, A., Anderson, A., Gregg, D. (2017). Parallel multi channel convolution using general matrix multiplication, *IEEE 28th international conference on application-specific systems, architectures and processors (ASAP)*, Seattle WA, ABD, 10-12 Temmuz 2017.

Véstias, M., Policarpo Duarte, R., T. de Sousa, J., Neto, H. (2018). Lite-CNN: A High-Performance Architecture to Execute CNNs in Low Density FPGAs, *28th International Conference on Field Programmable Logic and Applications (FPL)*, Dublin, İrlanda, 27-31 Ağustos 2018.

Vipin, K., Fahmy, S. A. (2014). ZyCAP: Efficient partial reconfiguration management on the Xilinx Zynq, *IEEE Embedded Systems Letters*, *6*(3), 41-44. DOI:10.1109/LES.2014.2314390

Vogel, S., Liang, M., Guntoro, A., Stechele, W., Ascheid, G. (2018). Efficient hardware acceleration of CNNs using logarithmic data representation with arbitrary log-base, *Proceedings of the International Conference on Computer-Aided Design*, San Diego CA, ABD, 5-8 Kasım 2018.

Wang, J., Lou, Q., Zhang, X., Zhu, C., Lin, Y., Chen, D. (2018). Design Flow of Accelerating Hybrid Extremely Low Bit-width Neural Network in Embedded FPGA, *2018 28th international conference on Field Programmable Logic and Applications (FPL)*, Dublin, İrlanda, 27-31 Ağustos 2018.

Wang, P., Chen, Q., He, X., Cheng, J. (2020). Towards accurate post-training network quantization via bit-split and stitching, *Proceedings of the 37th International Conference on Machine Learning*, Sanal, 13-18 Temmuz 2020.

Wei, X., Yu, C. H., Zhang, P., Chen, Y., Wang, Y., Hu, H., Liang, Y., Cong, J. (2017). Automated systolic array architecture synthesis for high throughput CNN inference on FPGAs, *Proceedings of the 54th Annual Design Automation Conference 2017*, Austin, ABD, 18-22 Haziran 2017.

Xu, J., Huan, Y., Jin, Y., Chu, H., Zheng, L.-R., Zou, Z. (2020). Base-reconfigurable segmented logarithmic quantization and hardware design for deep neural networks, *Journal of Signal Processing Systems.*, *92*(11), 1263-1276. DOI:10.1007/s11265-020-01557-8

Yan, Z., Shi, Y., Wang, Y., Tan, M., Li, Z., Tan, W., Tian, Y. (2020). Towards accurate low bit-width quantization with multiple phase adaptations, *Proceedings of the AAAI Conference on Artificial Intelligence*, C. 34 New York, ABD, 7-12 Şubat 2020.

Zhang, C., Li, P., Sun, G., Guan, Y., Xiao, B., Cong, J. (2015). Optimizing fpga-based accelerator design for deep convolutional neural networks, *Proceedings of the 2015 ACM/SIGDA international symposium on field-programmable gate arrays*, Monterey, ABD, 22-24 Şubat 2015.

Zhang, J., Li, J. (2017). Improving the performance of OpenCL-based FPGA accelerator for convolutional neural network, *Proceedings of the 2017 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays*, Monterey, ABD, 22-24 Şubat 2017.

Zhang, W., Jiang, M., Luo, G. (2020). Evaluating Low-Memory GEMMs for Convolutional Neural Network Inference on FPGAs, *IEEE 28th Annual International Symposium on Field-Programmable Custom Computing Machines (FCCM)*, Fayetteville AR, ABD, 3-6 Mayıs 2020.

Zhao, R., Hu, Y., Dotzel, J., De Sa, C., Zhang, Z. (2019). Improving neural network quantization without retraining using outlier channel splitting, *Proceedings of the 36th International Conference on Machine Learning*, Long Beach, ABD, 9-15 Haziran 2019.

Zhao, W., Fu, H., Luk, W., Yu, T., Wang, S., Feng, B., Ma, Y., Yang, G. (2016). F-CNN: An FPGA-based framework for training convolutional neural networks, *2016 IEEE 27Th international conference on application-specific systems, architectures and processors (ASAP)*, Londra, UK, 6-8 Temmuz 2016.

KİŞİSEL YAYINLAR VE ESERLER

Madadum, H., Becerikli, Y. (2017). The Implementation of Support Vector Machine (SVM) Using FPGA for Human Detection, *10th International Conference on Electrical and Electronics Engineering*, Bursa, Türkiye, 30 Kasım-2 Aralık 2017

Madadum, H., Becerikli, Y. (2019). FPGA-Based Optimized Convolutional Neural Network Framework for Handwritten Digit Recognition, *1st International Informatics and Software Engineering Conference*, Ankara, Türkiye, 6-7 Kasım 2019

Madadum, H., Becerikli, Y. (2022). A Resource-Efficient Convolutional Neural Network Accelerator Using Fine-Grained Logarithmic Quantization, *Intelligent Automation & Soft Computing*, 33(2), 681-695, DOI: 0.32604/iasc.2022.023831.

ÖZGEÇMİŞ

Hadee Madadum Bilgisayar Mühendisliği lisans öğrenimi 2011 yılında ve yüksek lisans öğrenimi 2015 yılında Tayland'da Prince of Songkla Üniversitesi'nden tamamladı. Halen Kocaeli Üniversitesi, İzmit, Türkiye'de Bilgisayar Mühendisliği Bölümü'nde doktora öğrenime devam etmektedir. İlgi araştırma alanları arasında FPGA tabanlı tasarım, gömülü sistemler, bilgisayar mimarileri ve derin sinir ağları bulunmaktadır.