T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

UÇ CİHAZLARDA DERİN ÖĞRENME TABANLI GÖRÜNTÜ SINIFLANDIRMADA MODEL OPTİMİZASYONU VE ENERJİ VERİMLİLİĞİ

YÜKSEK LİSANS

Asım Bilal YILMAZ

Bilişim Sistemleri Mühendisliği Anabilim Dalı

TEMMUZ 2025

T.C.
SAKARYA ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

UÇ CİHAZLARDA DERİN ÖĞRENME TABANLI GÖRÜNTÜ SINIFLANDIRMADA MODEL OPTİMİZASYONU VE ENERJİ VERİMLİLİĞİ

YÜKSEK LİSANS

Asım Bilal YILMAZ

Bilişim Sistemleri Mühendisliği Anabilim Dalı

Tez Danışmanı: Dr. Öğr. Üyesi Fatih ÇALLI

TEMMUZ 2025

Asım Bilal YILMAZ tarafından hazırlanan “Uç Cihazlarda Derin Öğrenme Tabanlı Görüntü Sınıflandırmada Model Optimizasyonu Ve Enerji Verimliliği” adlı tez çalışması 03.07.2025 tarihinde aşağıdaki jüri tarafından oy birliği/oy çokluğu ile Sakarya Üniversitesi Fen Bilimleri Enstitüsü Bilişim Sistemleri Mühendisliği Anabilim Dalı Yüksek Lisans tezi olarak kabul edilmiştir.

Tez Jürisi

Jüri Başkanı :

Jüri Üyesi :

Jüri Üyesi :

ETİK İLKE VE KURALLARA UYGUNLUK BEYANNAMESİ

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

(17/07/2025).

Asım Bilal YILMAZ

Eşime ve aileme,

TEŞEKKÜR

Yüksek lisans eğitimim boyunca ve tez çalışmamın her aşamasında yönlendirmeleri, tez konumu belirlemede ve çalışmalarım sırasında fikirlerini ve bilgilerini paylaşan ve tavsiyeleri ile bana yön veren değerli danışman hocam Dr. Öğr. Üyesi Fatih ÇALLI ‘ya sonsuz teşekkürler.

Ayrıca yaşamım boyunca arkamda duran, eğitim hayatımda kendilerinden aldığım destek ile bir adım ileriye gitmekte güç bulduğum tüm aile bireylerime özel teşekkürü bir borç bilirim.

Asım Bilal YILMAZ

İÇİNDEKİLER

Sayfa

KISALTMALAR

API : Uygulama Programlama Arayüzü
CIFAR : Kanada İleri Araştırma Enstitüsü
CNN : Evrişimli Sinir Ağı
DÖ : Derin Öğrenme
GPU : Grafik İşlem Birimi
ImageNet : ImageNet Büyük Ölçekli Görsel Tanıma
LSTM : Uzun Kısa Süreli Bellek
ML : Makine Öğrenmesi
NAS : Sinir Ağı Mimarisi Arama
NPU : Sinirsel İşlem Birimi
PTQ : Eğitim Sonrası Niceleme
QAT : Niceleme Farkındalıklı Eğitim
ReLU : Doğrultulmuş Lineer Ünite
RNN : Tekrarlayan Sinir Ağı
SDK : Yazılım Geliştirme Kiti
TPU : Tensor İşlem Birimi
YZ : Yapay Zeka

TABLO LİSTESİ

Sayfa

ŞEKİL LİSTESİ

Sayfa

UÇ CİHAZLARDADERİN ÖĞRENME TABANLI GÖRÜNTÜ SINIFLANDIRMADA MODEL OPTİMİZASYONU VE ENERJİ VERİMLİLİĞİ

ÖZET

Bu çalışma, derin öğrenme tekniklerinin uç cihazlardaki çeşitli uygulamalarını performans ve enerji verimliliği odağında detaylı bir şekilde analiz ederek alandaki bilgi birikimine önemli bir katkı sunmaktadır. Derin öğrenme algoritmalarındaki son yıllardaki dikkate değer ilerlemeler, bu algoritmaların bir araya gelmesiyle ortaya çıkan modellerin hem yüksek hesaplama gücü gereksinimleri hem de yüksek enerji tüketimi sorununu beraberinde getirmiştir. Bu durum, enerji verimliliği konusunda daha fazla araştırma ve geliştirme yapılmasını gerektiren bir durum haline gelmiştir.

Çalışma kapsamında, derin öğrenme modellerinin uç cihazlarda verimli bir şekilde çalışabilmesi için model sıkıştırma teknikleri üzerine kapsamlı ve detaylı birçok deney gerçekleştirilmiştir. Bu deneyler, özellikle budama ve nicemleme yöntemleri ile hafif mimarilerin entegrasyonu üzerine derinlemesine bir odaklanma sağlamaktadır.

Mobil cihazlardan akıllı sensörlere kadar geniş bir yelpazede yaygın olarak kullanılan MobileNet, EfficientNet ve ShuffleNet gibi hafif ve çağdaş mimarilerin entegrasyonu üzerinde durulmaktadır. Bu önemli çalışmalar, AMD NPU, Google Coral TPU ve NVIDIA Jetson Nano gibi çeşitli donanımlarla sistematik ve detaylı testler yapılarak yürütülmüştür.

Her bir test, belirli kriterlere göre özenle planlanmış ve uygulanmış, sonuçlar ise titizlikle analiz edilmiştir. Elde edilen en iyi performans sonuçları, güvenilir verilerle desteklenmekte ve derin öğrenme alanında yeni standartların belirlenmesine büyük katkı sağlamaktadır.

Çalışma kapsamda elde edilen bulgular, tez ile birlikte derin öğrenme teknolojilerinin özellikle uç cihazlarda daha erişilebilir hale gelmesi ve verimliliğinin önemli ölçüde artması için kritik bir zemin hazırlamaktadır. Derin öğrenme alanındaki yeni gelişmeler, mobil cihazlardan akıllı sensörlere kadar geniş bir uygulama yelpazesine sahip olup, bu da geleceğin teknolojilerini şekillendirme potansiyelini taşımaktadır. Bu sebeple, bu tez yalnızca mevcut uygulamaları detaylı bir şekilde analiz etmekle kalmamakta, aynı zamanda ileride gerçekleştirilecek araştırmalara ve geliştirme süreçlerine önemli katkılar sunmaktadır.

Çalışma, derin öğrenme tekniklerinin ve uygulamalarının gelişimini destekleyen kritik bir kaynak olma niteliğine sahiptir. Uç cihazlarda yapay zeka uygulamalarının etkinliğini artırmaya yönelik olarak ortaya konan bulgular, mühendisler ve akademisyenlerin iş birliğiyle daha ileri seviyelere taşınabilir. Nihayetinde, bu araştırma, mevcut bilgileri derlemenin ötesine geçerek, derin öğrenmenin geleceğine dair bir yol haritası sunmakta ve aynı zamanda sistemin genişlemesi için sağlam bir temel oluşturmaktadır. Bu durum, ileri düzeydeki çalışmaların zeminini hazırlar ve aynı zamanda endüstrinin ihtiyaçlarını karşılama konusundaki yetenekleri geliştirmeye katkı yapmaktadır.

Çalışmanın sağladığı içgörüler, derin öğrenmenin yenilikçi uygulamalarının teşvik edilmesine yardımcı olmakta ve bu sayede teknolojinin evrimi üzerinde kalıcı bir etki bırakmaktadır. Sonuç olarak bu araştırma, mevcut bilgileri derlemenin ötesine geçerek, derin öğrenmenin geleceğine dair bir yol haritası ve aynı zamanda sistemin ekosisteminin genişlemesi için sağlam bir temel sunmaktadır.

Anahtar Kelimeler: Derin Öğrenme, Görüntü Sınıflandırma, Uç Cihazlar, Model Optimizasyonu, Budama, Niceleme, MobileNet, EfficientNet, ShuffleNet, AMD NPU, Google Coral TPU, NVIDIA Jetson Nano, Enerji Verimliliği.

MODEL OPTIMISATION AND ENERGY EFFICIENCY IN DEEP LEARNING BASED IMAGE CLASSIFICATION ON EDGE DEVICES

SUMMARY

This paper makes a significant contribution to the body of knowledge in the field by analyzing in detail the various applications of deep learning techniques on edge devices with a focus on performance and energy efficiency. The remarkable advances in deep learning algorithms in recent years have led to both high computational power requirements and high energy consumption of the resulting models. This has necessitated further research and development on energy efficiency.

The focus is primarily on the integration of cutting-edge lightweight and contemporary architectures, notably MobileNet, EfficientNet, and ShuffleNet. These architectures are extensively utilized across a broad spectrum of applications, extending from mobile devices to innovative smart sensors that interact with various environments. This crucial work, characterized by its significance, has been meticulously carried out through systematic and comprehensive testing.

The hardware employed in this process is notably diverse, encompassing platforms such as the AMD NPU, Google Coral TPU, and NVIDIA Jetson Nano, each contributing to the validation and effectiveness of the architectures.

Each test was carefully planned and executed according to specific criteria, and the results were meticulously analyzed. The best performance results are supported by reliable data and contribute to setting new standards in the field of deep learning.

The findings of this study, combined with this thesis, establish a vital groundwork for deep learning technologies to become increasingly accessible, particularly on edge devices, while also significantly enhancing their efficiency. Recent developments within the field of deep learning have an extensive array of applications, extending from mobile devices to intelligent sensors, which possess the tremendous potential to influence and shape the technologies of the future.

Therefore, this thesis not only conducts a comprehensive analysis of existing applications in great detail but also offers essential contributions to future research and development processes, paving the way for innovations that could transform numerous industries and improve human-computer interactions in unprecedented ways.

The study serves as a vital resource that supports the ongoing development and enhancement of deep learning techniques and their various applications across multiple fields. The significant findings derived from this study can certainly be advanced further through collaborative efforts that unite engineers and academics strategically.

Their teamwork will greatly increase the overall effectiveness of AI applications, particularly on edge devices, which are increasingly important in today's technology ecosystem. Ultimately, this comprehensive research not only compiles the existing body of knowledge but also charts a well-defined roadmap for the future trajectory of

deep learning. Additionally, it lays a sturdy foundation that is crucial for the further expansion of this dynamic field.

This progressive approach paves the way for subsequent studies and contributes significantly to refining capabilities that are essential for meeting the evolving needs of the industry as a whole.

The insights provided by the study help foster innovative applications of deep learning, thereby leaving a lasting impact on the evolution of the technology. As a result, this research goes beyond compiling existing knowledge and provides a roadmap for the future of deep learning, as well as a solid foundation for the expansion of the system's ecosystem.

In this comprehensive and thorough study, a series of carefully designed and detailed experiments were meticulously conducted, focusing on advanced model compression techniques such as pruning and quantization.

Furthermore, the comprehensive and in-depth integration of a broad array of innovative lightweight architectures, which prominently include MobileNet, EfficientNet, and ShuffleNet, was explored in great detail. These targeted and specialized methodologies were specifically designed to guarantee not only the efficient operation of deep learning models but also to ensure high performance when they are deployed across a diverse range of devices. This, in turn, significantly enhances their usability and effectiveness across multiple platforms, making them suitable for various applications and use cases. The exploration of these architectures highlights their importance in the evolving landscape of technology.

Through systematic and rigorous testing across a diverse yet representative array of hardware platforms, including but not limited to AMD NPU, Google Coral TPU, and NVIDIA Jetson Nano, the researchers were able to ascertain the best performance results achievable with these methods.

This thorough process led to the collection of reliable and impactful data that underscores the effectiveness of the findings. By undertaking this extensive and in-depth research effort, the thesis makes a significant contribution to enhancing both the accessibility and efficiency of cutting-edge deep learning technologies. This advancement, in turn, facilitates their application on edge devices in a more effective and scalable manner, ultimately promoting a wider adoption of such technologies across various sectors.

Innovations in deep learning possess impressive and transformative potential to profoundly influence the future landscape of numerous advanced technologies, spanning from cutting-edge mobile devices designed for seamless user experiences to complex and sophisticated smart sensors that enhance various applications in our daily lives.

Given this significant impact, this thesis not only delivers a comprehensive analysis of existing applications and their various implications but also illuminates the promising avenues for future research and continuous development that lie ahead in this dynamic and exciting field.

By exploring these avenues, we can better understand the transformative power of deep learning and the opportunities it presents for innovation across multiple domains.

This work is a critical resource supporting the development of deep learning techniques and applications. These findings can be further developed with the

collaboration of engineers and academics to increase the effectiveness of AI applications on edge devices. In conclusion, this research goes beyond compiling existing knowledge and provides a roadmap for the future of deep learning.

Keywords: Deep Learning, Image Classification, Edge Devices, Model Optimization, Pruning, Quantization, MobileNet, EfficientNet, ShuffleNet, AMD NPU, Google Coral TPU, NVIDIA Jetson Nano, Energy Efficiency.

1. GİRİŞ

Günümüzde, dijital teknolojilerin hızla gelişmesi ve yaygınlaşmasıyla birlikte, veri miktarı ve çeşitliliği de katlanarak artmaktadır. Bu veri patlaması, geleneksel veri işleme yöntemlerinin yetersiz kalmasına ve yeni yaklaşımlara ihtiyaç duyulmasına yol açmıştır. Bu bağlamda, Yapay Zeka (YZ) ve Derin Öğrenme (DÖ), büyük ve karmaşık veri setlerinden anlamlı bilgiler çıkarma ve bu bilgileri çeşitli alanlarda karar verme süreçlerinde kullanma potansiyeli ile ön plana çıkmaktadır.

Derin öğrenme, yapay sinir ağlarının çok katmanlı yapısını kullanarak, veriden hiyerarşik özellikler öğrenmeyi ve karmaşık örüntüleri modellemeyi sağlayan bir makine öğrenmesi yöntemidir. Son yıllarda, derin öğrenme, görüntü sınıflandırma, nesne tanıma, doğal dil işleme ve konuşma tanıma gibi birçok alanda insan seviyesinde performans elde etmeyi başarmıştır.

Görüntü sınıflandırma, bilgisayarlı görü alanının temel problemlerinden biridir ve dijital görüntülerdeki nesnelerin veya sahnelerin otomatik olarak önceden tanımlanmış kategorilere atanması sürecini içerir. Derin öğrenme, özellikle Evrişimli Sinir Ağları (CNN) gibi mimariler aracılığıyla, görüntü sınıflandırma alanında önemli ilerlemeler kaydetmiştir. CNN ‘ler, görüntü verilerinden hiyerarşik özellikleri otomatik olarak öğrenme ve karmaşık örüntüleri tanıma yetenekleri sayesinde, geleneksel yöntemlere kıyasla çok daha yüksek doğruluk oranlarına ulaşabilmektedir. Ancak, derin öğrenme modellerinin yüksek hesaplama maliyetleri ve enerji tüketimleri, bu modellerin uç cihazlarda kullanılmasını zorlaştırmaktadır. Uç cihazlar, akıllı telefonlar, tabletler, giyilebilir cihazlar, sensörler gibi, verinin kaynağına yakın olan ve sınırlı işlem gücüne, bellek kapasitesine ve enerji kaynağına sahip olan cihazlardır. Bu cihazlarda derin öğrenme modellerini çalıştırmak, düşük gecikme süresi, yüksek enerji verimliliği ve gizlilik gibi gereksinimleri karşılamak açısından önemlidir.

Uç cihazlarda derin öğrenme, verinin kaynağında işlenmesini sağlayarak, bulut bilişime olan bağımlılığı azaltır, gecikme süresini düşürür, bant genişliği kullanımını optimize eder ve veri gizliliğini artırır. Bu nedenle, uç cihazlarda derin öğrenme

uygulamalarının geliştirilmesi, günümüzde büyük bir ilgi ve araştırma alanı haline gelmiştir.

1.1. Tezin Amacı

Bu tezin temel amacı, hafif ve verimli derin öğrenme modellerinin, farklı uç cihazlarda görüntü sınıflandırma performansı ve enerji verimliliği açısından kapsamlı bir analizini sunmaktır. Bu analiz, model sıkıştırma tekniklerinin (Budama ve Niceleme) ve farklı donanım platformlarının (AMD NPU, Google Coral TPU, NVIDIA Jetson Nano) etkilerini de içermektedir. Bu amaç doğrultusunda, aşağıdaki hedeflere ulaşılması planlanmaktadır:

Farklı uç cihazların (AMD NPU, Google Coral TPU, NVIDIA Jetson Nano) donanım özelliklerinin ve mimarilerinin, derin öğrenme modellerinin performansı üzerindeki etkisini incelemek.

Hafif derin öğrenme modellerinin (MobileNetV3, EfficientNet-B0, ShuffleNet V2) ve model sıkıştırma tekniklerinin (Budama ve Niceleme) uç cihazlardaki performans ve enerji verimliliğine etkisini değerlendirmek.

Farklı model mimarileri, sıkıştırma yöntemleri ve cihaz kombinasyonlarını karşılaştırarak, belirli bir doğruluk ve hız gereksinimi için en uygun çözümü belirlemek.

Uç cihazlarda derin öğrenme tabanlı görüntü sınıflandırma uygulamalarının geliştirilmesine yönelik pratik bilgiler ve öneriler sunmak.

1.2. Literatür Araştırması

Bu bölümde, tezin konusuyla ilgili olarak yapay zeka, derin öğrenme, görüntü sınıflandırma, uç cihazlarda derin öğrenme, model sıkıştırma teknikleri, hafif ve verimli model mimarileri ve spesifik uç cihazlar hakkında detaylı bir literatür taraması sunulacaktır. Ayrıca, bu alandaki mevcut çalışmalar ve yaklaşımlar incelenerek, tezin literatürdeki boşluğu ve özgün katkısı ortaya konulacaktır.

1.3. Tezin Kapsamı

Bu tez çalışması, üç farklı uç cihaz olarak bilinen teknoloji aygıtı (AMD NPU, Google Coral TPU, NVIDIA Jetson Nano) üzerinde üç farklı hafif derin öğrenme modelinin

(MobileNetV3, EfficientNet-B0, ShuffleNet V2) performans ve enerji verimliliği analizi üzerine kapsamlı ve detaylı bir inceleme sunmaktadır. Çalışma kapsamında, popüler veri setlerinden olan ImageNet (alt kümesi), CIFAR-10 ve CIFAR-100 kullanılmak suretiyle bu modeller, görüntü sınıflandırma görevi için kapsamlı bir şekilde test edilmiş ve dikkatlice eğitim sürecinden geçirilmiştir. Elde edilen analiz sonuçları, bu modellerin performansları ve enerji verimlilikleri açısından çeşitli uç cihazlar üzerindeki etkinliklerini ve işleyişlerini ortaya koyarak karşılaştırılabilir bir yapıda sunmaktadır. Bu çalışma, alanında önemli veriler sağlayarak, derin öğrenme uygulamalarının farklı cihazlar üzerinde ne denli etkili ve verimli olabileceğine dair değerli bilgiler içermektedir.

Deneyler sırasında, çeşitli modellerin doğruluk, çıkarım süresi, CPU kullanımı, RAM kullanımı ve enerji tüketimi gibi önemli performans metrikleri kapsamlı bir şekilde ölçülmüş ve karşılaştırılmıştır. Ayrıca, gerçekleştirilen araştırmalarda Budama (ağırlık ve filtre Budama) ile niceleme (PTQ ve QAT) tekniklerinin bu metrikler üzerindeki etkileri titizlikle analiz edilmiştir. Bu durum, modellerin genel performansı hakkında kapsamlı ve güvenilir verilere ulaşılmasını sağlamıştır.

Tez kapsamında, gerçek zamanlı nesne tanıma gibi farklı derin öğrenme görevleri veya video işleme gibi daha karmaşık uygulamalar ele alınmamıştır. Ayrıca, dağıtık öğrenme (federated learning) gibi konular da tez kapsamı dışında tutulmuştur.

1.4. Literatürdeki Boşluklar ve Tezin Katkısı

Derin öğrenme ve görüntü sınıflandırma alanında birçok çalışma yapılmış olmasına rağmen (Krizhevsky ve diğerleri, 2012; LeCun ve diğerleri, 2015; Simonyan ve Zisserman, 2014; He ve diğerleri, 2016; Huang ve diğerleri, 2017; Tan ve Le, 2019), bu çalışmaların çoğu, genellikle bulut tabanlı sistemlerde veya sunucularda, yüksek performanslı donanımlar üzerinde gerçekleştirilmiştir. Uç cihazlarda derin öğrenme modellerinin performansını ve enerji verimliliğini optimize etmeye yönelik çalışmalar ise henüz sınırlı sayıdadır (Han ve diğerleri, 2015; Howard ve diğerleri, 2017; Sandler ve diğerleri, 2018; Zhang ve diğerleri, 2018; Ma ve diğerleri, 2018).

Farklı uç cihazların karşılaştırmalı analizlerinin eksikliği: Çoğu çalışma, tek bir cihaz veya platform üzerinde yoğunlaşmaktadır. Bu tez, AMD NPU, Google Coral TPU ve NVIDIA Jetson Nano gibi farklı donanım mimarilerine sahip üç popüler uç cihazı karşılaştırmalı olarak inceleyerek bu boşluğu doldurmayı hedeflemektedir.

Model sıkıştırma tekniklerinin uç cihazlardaki etkilerinin kapsamlı bir şekilde değerlendirilmemesi: Literatürde, Budama ve niceleme gibi tekniklerin modellerin boyutu ve hızı üzerindeki etkileri incelenmiş olsa da, bu tekniklerin farklı uç cihazlardaki enerji tüketimi ve doğruluk üzerindeki etkileri yeterince araştırılmamıştır. Bu tez, farklı sıkıştırma tekniklerinin ve oranlarının, modellerin performans metrikleri üzerindeki etkilerini detaylı bir şekilde analiz ederek bu eksikliği gidermeyi amaçlamaktadır.

Gerçek dünya uygulamalarına yönelik pratik bilgilerin yetersizliği: Çoğu çalışma, teorik sonuçlara odaklanmakta ve gerçek dünya uygulamalarında karşılaşılabilecek zorlukları ve kısıtlamaları yeterince ele almamaktadır. Bu tez, uç cihazlarda derin öğrenme tabanlı görüntü sınıflandırma uygulamalarının geliştirilmesine yönelik pratik bilgiler ve öneriler sunarak bu boşluğu doldurmayı hedeflemektedir.

Farklı uç cihazlar (AMD NPU, Google Coral TPU, NVIDIA Jetson Nano) üzerinde, hafif ve verimli derin öğrenme modellerinin (MobileNetV3, EfficientNet-B0, ShuffleNetV2) kapsamlı bir performans ve enerji verimliliği analizini sunmaktadır.

Model sıkıştırma tekniklerinin (Budama ve niceleme) ve farklı uygulama yöntemlerinin (ağırlık Budama, filtre Budama, PTQ, QAT) uç cihazlardaki etkilerini detaylı bir şekilde incelemektedir.

Farklı model, sıkıştırma ve cihaz kombinasyonları için en uygun optimizasyon stratejilerini belirleyerek, alandaki araştırmacılara ve uygulayıcılara pratik bir rehber sunmaktadır.

Uç cihazlarda derin öğrenme uygulamalarının daha erişilebilir, verimli ve yaygın hale gelmesine katkı sağlayarak, bu alandaki literatür boşluğunu doldurmaktadır.

2. LİTERATÜR TARAMASI

2.1. Yapay Zeka Tarihçesi

Yapay zeka, insanların düşünme ve problem çözme becerilerini taklit eden sistemlerin geliştirilmesiyle ilgilenen, oldukça ilgi çekici bir bilim dalıdır. Bugün geldiğimiz noktada, yapay zeka hayatımızın her alanında yer almaktadır. Yapay zeka alandaki çalışmalar aslında 20. yüzyılın ortalarına dayanmaktadır.

Her şey, Alan Turing'in "Makineler düşünebilir mi?" sorusuyla başladı. 1950'de Turing, bu sorunun cevabını bulabilmek için Turing Testi olarak bilinen bir yöntem önerdi. Bu test, bir makinenin insan gibi "düşünebildiğini" anlamak için hala önemli bir referans noktasıdır (Turing, 1950). Turing'in bu öncü çalışmaları, o dönemin bilim dünyasında büyük bir yankı uyandırdı ve 1956'da yapılan Dartmouth Konferansı, yapay zekayı bağımsız bir araştırma alanı olarak resmen başlattı. Bu konferans sırasında, John McCarthy ve diğer bilim insanları "yapay zeka" terimini ilk kez kullanarak, bugün bildiğimiz YZ alanının temellerini attılar.

İlk dönemlerde, yapay zekanın temeli "sembolik yaklaşım" üzerine kuruluydu. Bu yaklaşıma göre, bilgi sembollerle temsil ediliyor ve problemler mantıksal çıkarım kurallarıyla çözülüyordu. Mesela, Genel Problem Çözücü (GPS) (Newell ve Simon, 1963) ve bir tür erken dönem sohbet botu olan ELIZA (Weizenbaum, 1966), bu yaklaşımın önemli örnekleri arasında sayılabilir. Ancak, bu sistemler daha karmaşık problemleri çözmekte zorlanıyordu.

1980'lere geldiğimizde, sahneye "uzman sistemler" çıktı. Bu sistemler, bir alandaki uzmanların bilgilerini kullanarak sorunları çözmeye odaklanıyordu. Örneğin, MYCIN (Shortliffe, 1976) enfeksiyon hastalıklarını teşhis ederken oldukça etkiliydi. Benzer şekilde, DENDRAL kimya alanında, PROSPECTOR ise jeoloji alanında başarılı uygulamalara sahipti. Ancak, uzman sistemler de sınırlı veriyle çalışıyor ve gerçek dünyadaki belirsizlikleri modellemekte zorlanıyordu.

1990'larda ise makine öğrenmesi (ML) sahneye çıktı ve adeta yapay zekanın çehresini değiştirdi. Makine öğrenmesi, sistemlerin verilerden öğrenmesini ve tahmin yapmasını mümkün kılıyordu. Bu dönemde geliştirilen destek vektör makineleri (SVM) (Cortes

ve Vapnik, 1995), karar ağaçları (Quinlan, 1986) ve rastgele ormanlar (Breiman, 2001) gibi algoritmalar, makine öğrenmesinin önemli adımlarıydı. Bu teknikler, daha fazla veri ve daha güçlü hesaplama kaynakları sayesinde hızla popülerlik kazandı.

2000'li yıllarda ise yapay zekada büyük gelişmeler yaşandı. Derin öğrenme (Deep Learning), yapay sinir ağlarının daha büyük ve derin yapılarını kullanarak, büyük veri kümelerinden karmaşık ilişkiler öğrenebilmeyi mümkün kıldı. Convolutional Neural Networks (CNNs) gibi modeller, özellikle görüntü işleme alanında çığır açtı (Krizhevsky ve diğerleri, 2012). Transformers gibi modeller ise doğal dil işleme alanında (örneğin BERT ve GPT gibi) devrim yarattı (Vaswani ve diğerleri, 2017).

Bugün yapay zeka sadece akademik bir alan değil, aynı zamanda otonom araçlardan sağlık teknolojilerine, dijital asistanlardan enerji optimizasyonuna kadar sayısız sektörü dönüştüren bir güç haline gelmiş durumda. Gelecek yıllarda kuantum hesaplama ve daha sürdürülebilir yapay zeka sistemleri gibi konuların bu alanda belirleyici olması bekleniyor.

2.2. Derin Öğrenmeye Giriş

Derin öğrenme, özellikle son yıllarda görüntü işleme (Krizhevsky ve diğerleri, 2012), doğal dil işleme (Vaswani ve diğerleri, 2017) ve konuşma tanıma (Hinton ve diğerleri, 2012) gibi alanlarda önemli başarılara imza atmıştır.

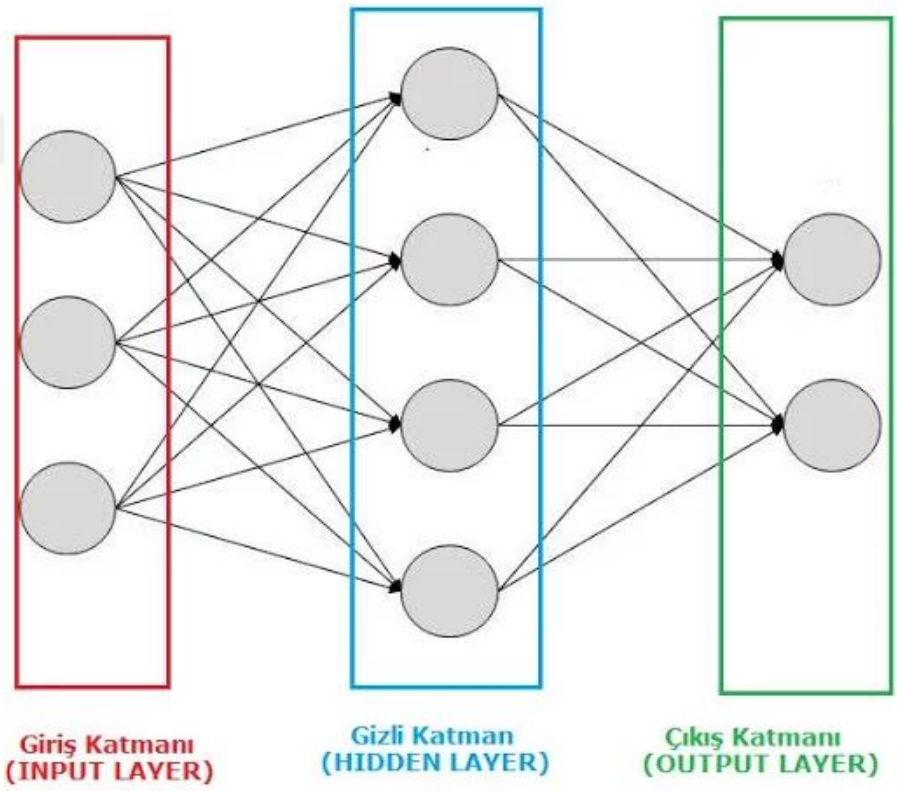
Derin öğrenmenin temelini oluşturan yapay sinir ağları (YSA), insan beyninin yapısından esinlenerek geliştirilmiş matematiksel modellerdir. YSA'lar, birbirine bağlı yapay nöronlardan oluşur ve her nöron, girdi olarak aldığı bilgileri işleyerek bir çıktı üretir. Nöronlar arasındaki bağlantıların ağırlıkları, eğitim sürecinde ayarlanarak, ağın belirli bir görevi yerine getirmesi sağlanır. Yapay sinir ağı mimarisi örneği Şekil 2.1'de gösterilmiştir. Derin öğrenme modelleri, çok sayıda gizli katmana sahip olmalarıyla geleneksel YSA'lardan ayrılır. Bu sayede, veriden daha karmaşık ve soyut özellikleri öğrenebilirler. Derin öğrenme modellerinin eğitimi, genellikle büyük miktarda veri ve yüksek hesaplama gücü gerektirir.

1990'larda ise makine öğrenmesi (ML) sahneye çıktı ve adeta yapay zekanın çehresini değiştirdi. Makine öğrenmesi, sistemlerin verilerden öğrenmesini ve tahmin yapmasını mümkün kılıyordu. Bu dönemde geliştirilen destek vektör makineleri (SVM) (Cortes ve Vapnik, 1995), karar ağaçları (Quinlan, 1986) ve rastgele ormanlar (Breiman, 2001)

gibi algoritmalar, makine öğrenmesinin önemli adımlarıydı. Bu teknikler, daha fazla veri ve daha güçlü hesaplama kaynakları sayesinde hızla popülerlik kazandı.

2000'li yıllarda ise yapay zekada büyük gelişmeler yaşandı. Derin öğrenme (Deep Learning), yapay sinir ağlarının daha büyük ve derin yapılarını kullanarak, büyük veri kümelerinden karmaşık ilişkiler öğrenebilmeyi mümkün kıldı. Convolutional Neural Networks (CNNs) gibi modeller, özellikle görüntü işleme alanında çığır açtı (Krizhevsky ve diğerleri, 2012). Transformers gibi modeller ise doğal dil işleme alanında (örneğin BERT ve GPT gibi) devrim yarattı (Vaswani ve diğerleri, 2017).

Bugün yapay zeka sadece akademik bir alan değil, aynı zamanda otonom araçlardan sağlık teknolojilerine, dijital asistanlardan enerji optimizasyonuna kadar sayısız sektörü dönüştüren bir güç haline gelmiş durumda. Gelecek yıllarda kuantum hesaplama ve daha sürdürülebilir yapay zeka sistemleri gibi konuların bu alanda belirleyici olması bekleniyor.

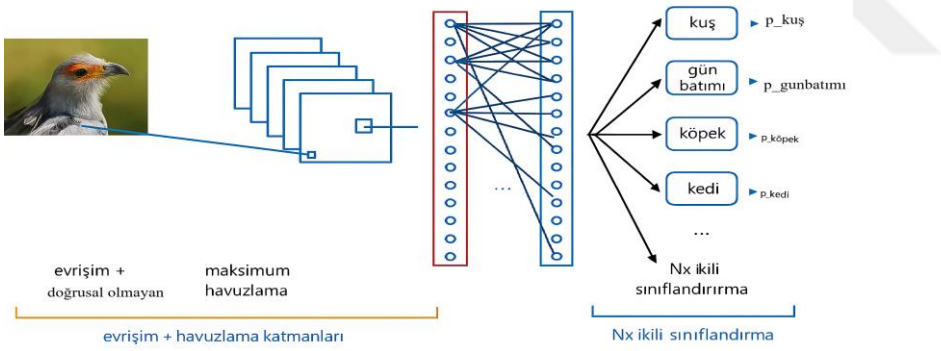


Şekil 2.1. Yapay Sinir Ağı Mimarisi Örneği (Kaynak: Elmas, 2021).

2.3. Görüntü Sınıflandırma ve Derin Öğrenme

Görüntü sınıflandırma, bilgisayarlı görü alanının temel problemlerinden biridir ve bir görüntüdeki nesnelerin veya sahnelerin otomatik olarak önceden tanımlanmış kategorilere atanması sürecini içerir. Derin öğrenme, özellikle Evrişimli Sinir Ağları (CNN) gibi mimariler aracılığıyla, görüntü sınıflandırma alanında önemli ilerlemeler kaydetmiştir (LeCun ve diğerleri, 2015).

CNN'ler, görüntü verilerinden hiyerarşik özellikleri otomatik olarak öğrenmek için evrişim (convolution), havuzlama (pooling) ve aktivasyon katmanları gibi özel katman türleri kullanır. Evrişimli sinir ağı katmanları görüntüsü Şekil 2.2 de gösterilmiştir. Evrişim katmanları, görüntüdeki yerel bölgelerden özellikler çıkarmak için filtreler (çekirdekler) kullanır. Farklı filtreler, kenarlar, köşeler ve dokular gibi farklı özellikleri tespit eder. Havuzlama katmanları, özellik haritalarının boyutunu küçülterek hesaplama maliyetini azaltır ve modelin öteleme değişmezliğine katkı sağlar. Aktivasyon katmanları ise, sinir ağlarına doğrusal olmayanlık kazandırarak, modelin daha karmaşık işlevleri öğrenmesini sağlar.



Şekil 2.2. Evrişimli Sinir Ağı Katmanları. (Kaynak: Krizhevsky, 2009).

Bu mimariler, ImageNet (Deng ve diğerleri, 2009), CIFAR-10 ve CIFAR-100 (Krizhevsky, 2009) gibi büyük ölçekli veri setleri üzerinde eğitilerek, insan seviyesinde performans elde etmeyi başarmıştır. LeNet (LeCun ve diğerleri, 1998), AlexNet (Krizhevsky ve diğerleri, 2012), VGGNet (Simonyan ve Zisserman, 2014), GoogLeNet (Szegedy ve diğerleri, 2015) ve ResNet (He ve diğerleri, 2016) gibi CNN mimarileri, görüntü sınıflandırma alanında çığır açan gelişmelere öncülük etmiştir.

Derin öğrenme modellerinin bulut tabanlı sistemlerden uç cihazlara (mobil cihazlar, IoT cihazları, gömülü sistemler vb.) taşınması, günümüzün önemli teknoloji

trendlerinden biridir (Lane ve diğerleri, 2015). Uç cihazlarda derin öğrenme, verinin toplandığı yerde işlenmesi sayesinde düşük gecikme (low latency), yüksek gizlilik (privacy), azaltılmış bant genişliği kullanımı (reduced bandwidth usage) ve bulut tabanlı sistemlere olan bağımlılığın azaltılması gibi avantajlar sunmaktadır (Satyanarayanan, 2017).

Ancak, uç cihazların sınırlı hesaplama gücü, bellek ve enerji kaynakları, derin öğrenme modellerinin bu cihazlarda etkin bir şekilde çalışmasını zorlaştırmaktadır (Han ve diğerleri, 2015). Bu nedenle, uç cihazlarda çalışabilecek, enerji verimli ve hızlı derin öğrenme modellerine olan ihtiyaç, model sıkıştırma (pruning, quantization) (Han ve diğerleri, 2015; Cheng ve diğerleri, 2018), model hızlandırma (Chen ve diğerleri, 2018; Iandola ve diğerleri, 2016) ve enerji verimliliği odaklı çeşitli tekniklerin geliştirilmesine yol açmıştır (Howard ve diğerleri, 2017).

Uç cihazlar, donanımsal ve yazılımsal kısıtlamaları nedeniyle derin öğrenme modellerinin uygulanması açısından bazı zorluklarla karşı karşıyadır. Bu zorluklar şunlardır:

Sınırlı Hesaplama Kapasitesi: Uç cihazlar genellikle düşük güçlü işlemcilere sahiptir ve bu nedenle karmaşık derin öğrenme modellerini çalıştırmak için yeterli hesaplama kapasitesine sahip değildir.

Kısıtlı Bellek Kaynakları: Derin öğrenme modelleri, büyük miktarda bellek gerektirir. Uç cihazlar ise sınırlı bellek kapasitesine sahiptir.

Enerji Tüketimi Sınırlamaları: Uç cihazlar genellikle pil ile çalışır ve bu nedenle enerji tüketiminin minimize edilmesi önemlidir.

Gerçek Zamanlı Performans Gereksinimleri: Birçok uç cihaz uygulamasında, düşük gecikme süresi ve gerçek zamanlı sonuç üretme gereksinimi vardır.

Uç cihazlarda derin öğrenme modellerinin etkin bir şekilde uygulanabilmesi için, bu zorlukların üstesinden gelinmesi ve belirli gereksinimlerin karşılanması gerekmektedir. Bu gereksinimler şunlardır:

Kompakt Model Boyutu: Uç cihazların sınırlı bellek kapasitesi göz önünde bulundurularak, modellerin kompakt bir boyuta sahip olması gerekir.

Düşük Hesaplama Maliyeti: Modellerin, düşük güçlü işlemcilerde bile hızlı ve verimli bir şekilde çalışabilmesi için düşük hesaplama maliyetine sahip olması gerekir.

Enerji Verimliliği: Modellerin, minimum enerji tüketerek maksimum performans sağlaması gerekir.

Düşük Gecikme Süresi: Modellerin, gerçek zamanlı uygulamalarda kullanılabilmesi için düşük gecikme süresine sahip olması gerekir.

Yüksek Doğruluk Oranı: Tüm bu kısıtlamalara rağmen, modellerin yüksek doğruluk oranını koruması gerekir.

2.4. Model Sıkıştırma Teknikleri:Prunning ve Quantization

Uç cihazlarda derin öğrenme modellerinin verimli ve etkili bir şekilde çalışabilmesi için, bu modellerin performansını artırmak amacıyla model sıkıştırma teknikleri yaygın olarak kullanılmaktadır. Bu önemli teknikler, modellerin boyutunu, hesaplama karmaşıklığını azaltmayı ve enerji tüketimini minimum seviyeye indirmeyi hedefler. En yaygın olarak kullanılan model sıkıştırma teknikleri arasında, özellikle pruning (Budama) ve quantization (niceleme) gibi yöntemler öne çıkmaktadır. Bu teknikler, derin öğrenme uygulamalarının daha verimli bir şekilde çalışmasını sağlamaktadır.

2.4.1. Pruning (Budama)

Pruning, derin öğrenme modellerindeki gereksiz bağlantıların veya nöronların kaldırılması işlemidir. Bu sayede, modelin boyutu küçülür, hesaplama karmaşıklığı azalır ve enerji verimliliği artar. Pruning, ağırlık Budama (weight pruning) ve filtre Budama (filter pruning) olmak üzere iki ana kategoriye ayrılabilir (Li ve diğerleri, 2016; Molchanov ve diğerleri, 2017).

2.4.1.1. Ağırlık budama (Weight pruning)

Ağırlık Budama, modeldeki bireysel ağırlıkların (bağlantıların) sıfıra çekilerek modelin seyreltilmesi işlemidir. Ağırlık Budama, modeldeki bağlantıların bir kısmını kaldırarak modelin boyutunu ve bellek kullanımını azaltır. Farklı ağırlık Budama yöntemleri arasında, büyüklüğe dayalı Budama (magnitude-based pruning) (Han ve diğerleri, 2015), gradyan tabanlı Budama (gradient-based pruning) (LeCun ve diğerleri, 1990) ve Hessian tabanlı Budama (Hessian-based pruning) (LeCun ve diğerleri, 1990) gibi teknikler bulunmaktadır.

2.4.1.2. Filtre budama (Filter pruning)

Filtre Budama, modeldeki belirli filtrelerin veya kanalların tamamen kaldırılması işlemidir. Filtre Budama, ağırlık Budamaya göre daha kaba tanelidir (coarse-grained) ve modelin yapısını daha fazla değiştirir. Ancak, hesaplama maliyetini ve bellek kullanımını önemli ölçüde azaltabilir (Li ve diğerleri, 2016; He ve diğerleri, 2017; Luo ve diğerleri, 2017). Farklı filtre Budama yöntemleri arasında, L1 normuna dayalı Budama (Li ve diğerleri, 2016), geometrik medyan tabanlı Budama (He ve diğerleri, 2017) ve kanal önemine dayalı Budama (importance-based pruning) (Molchanov ve diğerleri, 2017) gibi teknikler bulunmaktadır.

2.4.1.3. Pruning algoritmaları

Pruning işlemini gerçekleştirmek için farklı algoritmalar ve yaklaşımlar kullanılmaktadır:

Gradient-Based Pruning: Ağırlıkların veya filtrelerin, modelin çıktıları üzerindeki etkisini tahmin etmek için gradyan bilgileri kullanılır ve buna göre Budama yapılır (LeCun ve diğerleri, 1990).

Sensitivity-Based Pruning: Modelin farklı katmanlarının Budamaya karşı hassasiyeti analiz edilerek, hangi katmanların daha fazla budanabileceği belirlenir (Han ve diğerleri, 2015).

L1/L2 Regularizasyonu: Modelin eğitim sürecine, ağırlıkların L1 veya L2 normunu içeren bir ceza terimi eklenerek, modelin daha seyrek ağırlıklara sahip olması teşvik edilir. L1 regularizasyonu, ağırlıkları doğrudan sıfıra çekerken, L2 regularizasyonu ağırlıkları sıfıra yaklaştırır (Li ve diğerleri, 2016; Molchanov ve diğerleri, 2017).

Importance-Based Pruning: Modeldeki ağırlıkların, filtrelerin veya kanalların önem dereceleri, belirli bir metrik (örneğin, ağırlık büyüklüğü, gradyan bilgisi) kullanılarak hesaplanır ve en az önemli olanlar budanır (Han ve diğerleri, 2015; Molchanov ve diğerleri, 2017).

Neural Architecture Search (NAS) Tabanlı Budama: NAS yöntemleri kullanılarak hem model mimarisinin hem de Budama oranlarının otomatik olarak optimize edilmesi sağlanır (Liu ve diğerleri, 2019).

2.4.2. Quantization (Niceleme)

Quantization, modelin ağırlıklarının ve aktivasyonlarının daha düşük bit derinliğine (örneğin, 32-bit float'tan 8-bit integer'a) dönüştürülmesi işlemidir. Bu sayede, modelin boyutu küçülür, bellek kullanımı azalır ve hesaplama hızı artar (Jacob ve diğerleri, 2018). Quantization, eğitim sonrası Niceleme (post-training quantization - PTQ) ve Niceleme farkındalıklı eğitim (quantization-aware training - QAT) olmak üzere iki ana kategoriye ayrılabilir.

2.4.2.1. Post-training quantization (PTQ)

Post-training quantization (PTQ), eğitilmiş bir modelin ağırlıklarının ve aktivasyonlarının, eğitimden sonra Nicelenmesi işlemidir. PTQ, uygulaması daha kolay ve hızlı bir yöntemdir, ancak QAT'a kıyasla doğruluk kaybına neden olabilir (Jacob ve diğerleri, 2018). PTQ kapsamında statik niceleme ve dinamik niceleme olmak üzere iki farklı yöntem vardır. Statik Niceleme yönteminde, kalibrasyon verisi kullanılarak, ağırlıkların ve aktivasyonların minimum ve maksimum değerleri belirlenir ve bu değerlere göre bir Niceleme aralığı hesaplanır. Dinamik Niceleme yönteminde ise, Niceleme parametreleri (örneğin, ölçek ve ofset değerleri) çalışma zamanında (runtime) hesaplanır ve her bir tensör için ayrı ayrı uygulanır.

2.4.2.2. Quantization-aware training (QAT)

Quantization-aware training (QAT) , modelin eğitim süreci sırasında Nicelemenin etkilerinin simüle edilerek, modelin Nicelenmiş ağırlıklara ve aktivasyonlara göre eğitilmesi işlemidir. QAT, genellikle PTQ'ya göre daha yüksek doğruluk sağlar, ancak daha fazla zaman ve hesaplama kaynağı gerektirir (Choi ve diğerleri, 2018).

2.4.2.3. Quantization şemaları

Farklı quantization şemaları ve algoritmaları bulunmaktadır:

Uniform Quantization: Ağırlıklar ve aktivasyonlar, eşit aralıklı değerlere dönüştürülür.

Non-Uniform Quantization: Ağırlıkların ve aktivasyonların dağılımına göre değişen aralıklar kullanılarak (örneğin, logaritmik ölçekleme) daha hassas bir niceleme yapılır.

Dinamik Quantization: Niceleme parametreleri (örneğin, ölçek ve ofset değerleri) çalışma zamanında (runtime) ayarlanarak daha iyi performans elde etmeyi amaçlar.

Ternary Quantization: Ağırlıklar ve aktivasyonlar sadece üç değere (-1, 0, +1) dönüştürülür (Zhu ve diğerleri, 2016).

Binary Quantization: Ağırlıklar ve aktivasyonlar sadece iki değere (-1, +1) dönüştürülür (Courbariaux ve diğerleri, 2016).

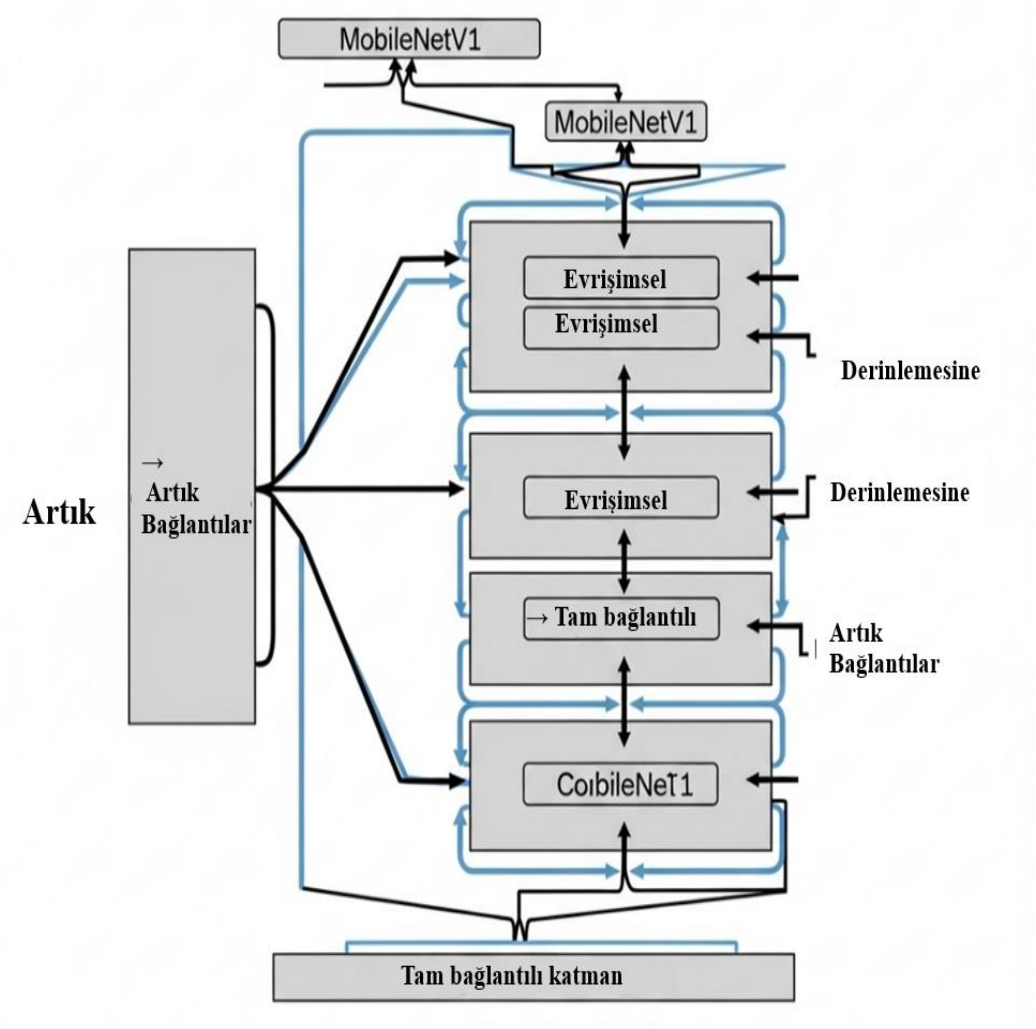
2.5. Hafif ve Verimli Model Mimarileri

Uç cihazlarda derin öğrenme modellerini çalıştırmak için, hafif ve verimli model mimarilerinin kullanılması önemlidir. Bu mimariler, daha az parametreye, daha düşük hesaplama karmaşıklığına ve daha az bellek kullanımına sahiptir. Son yıllarda, bu amaçla geliştirilmiş birçok hafif model mimarisi bulunmaktadır. Bu tez çalışmasında, MobileNet, EfficientNet ve ShuffleNet mimarileri incelenmiştir.

2.5.1. MobileNet

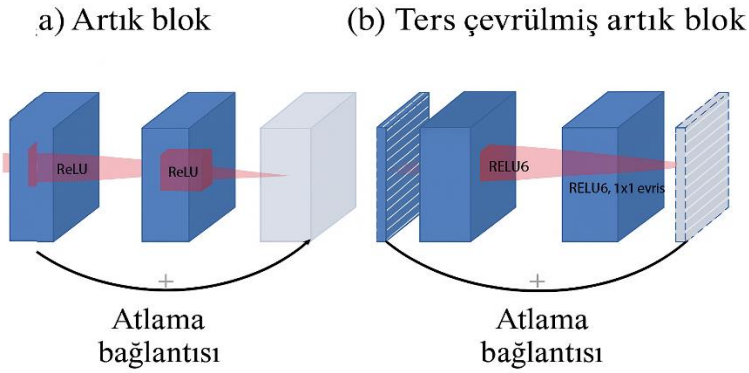
MobileNet (Howard ve diğerleri, 2017), derinlemesine ayrılabilir evrişim (depthwise separable convolution) adı verilen bir yapı taşı kullanılarak geliştirilmiş hafif bir CNN mimarisidir. Derinlemesine ayrılabilir evrişim, standart bir evrişimi, derinlemesine (depthwise) ve noktasal (pointwise) olmak üzere iki ayrı katmana ayırır. Derinlemesine evrişim, her bir girdi kanalına ayrı ayrı uygulanırken, noktasal evrişim, derinlemesine evrişim katmanından gelen çıktıları birleştirmek için kullanılır. Bu ayrım, hesaplama maliyetini ve modeldeki parametre sayısını önemli ölçüde azaltır.

MobileNetV1 (Howard ve diğerleri, 2017), MobileNet mimarisinin ilk versiyonudur ve derinlemesine ayrılabilir evrişim kullanarak, standart evrişim katmanlarına göre daha az parametre ve hesaplama gerektirir. BMobileNetV1 mimarisi Şekil 2.3'de verilmiştir.



Şekil 2.3. MobileNetV1 Mimarisi. (Kaynak: Howard ve diğerleri, 2017).

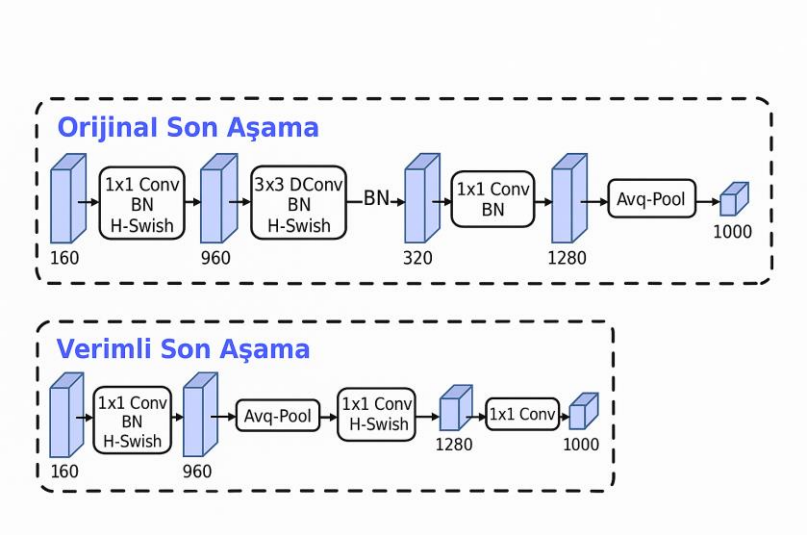
MobileNetV2 (Sandler ve diğerleri, 2018), ters çevrilmiş artık bloklar (inverted residual blocks) ve doğrusal darboğazlar (linear bottlenecks) gibi yeni yapısal bileşenler ekleyerek, MobileNetV1'in performansını ve verimliliğini artırmıştır.



Şekil 2.4. MobileNetV2 Mimarisi ve Ters Çevrilmiş Artık Blok Yapısı. (Kaynak: Sandler ve diğerleri, 2018).

Bu şekilde (a) Genişletme katmanı (1x1 evrişim), derinlemesine evrişim (3x3 derinlemesine ayrılabilir evrişim) ve çıkış katmanından (1x1 evrişim) oluşan ters çevrilmiş artık blok (inverted residual block) yapısı. (b) Adım (stride) 2 olduğunda ters çevrilmiş artık blok yapısı. Ters çevrilmiş artık bloklar, doğrusal darboğazlarla (linear bottlenecks) birleştirilerek, modelin daha az parametre ile daha yüksek doğruluk elde etmesini sağlar. Bu mimarinin uygulandığı görüntü sınıflarına ilişkin örnekler Şekil 3.1'de verilmiştir.

MobileNetV3 (Howard ve diğerleri, 2019) ise, mimari arama algoritmaları (NAS) kullanılarak otomatik olarak optimize edilmiş bir mimaridir ve platforma özgü optimizasyonlar ve h-swish aktivasyon fonksiyonu gibi yeni bileşenler içermektedir.



Şekil 2.5. MobileNetV3 Mimarisi. (Kaynak: Howard ve diğerleri, 2019).

Şekilde, MobileNetV3-Large ve MobileNetV3-Small modellerinin mimari detayları gösterilmektedir. Her iki model de derinlemesine ayrılabilir evrişim, ters çevrilmiş artık bloklar ve ağ mimarisi arama (NAS) ile optimize edilmiş yapı taşları içermektedir. Ayrıca, h-swish aktivasyon fonksiyonu ve squeeze-and-excitation blokları gibi yenilikler kullanılmıştır.

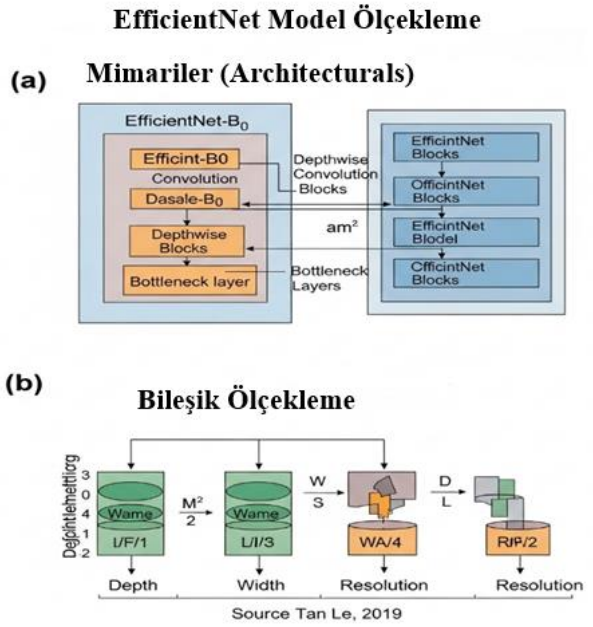
MobileNet modelleri, hafif olmaları ve düşük hesaplama gereksinimleri nedeniyle, uç cihazlarda kullanım için oldukça uygundur.

2.5.2. EfficientNet

EfficientNet (Tan ve Le, 2019), bileşik ölçeklendirme (compound scaling) adı verilen bir yöntem kullanarak, modelin derinliğini, genişliğini ve çözünürlüğünü dengeli bir şekilde ölçeklendiren bir CNN mimarisidir. Bileşik ölçeklendirme, modelin tüm

boyutlarının (derinlik, genişlik, çözünürlük) belirli bir katsayı ile çarpılarak büyütülmesi veya küçültülmesi işlemidir. Bu yöntem, modelin performansını ve verimliliğini optimize etmek için, her bir boyutun nasıl ölçeklendirileceğini belirler.

EfficientNet-B0, EfficientNet ailesinin temel modelidir ve bileşik ölçeklendirme yöntemi kullanılarak, en iyi doğruluk ve verimlilik dengesini sağlayacak şekilde tasarlanmıştır.



Şekil 2.6. EfficientNet Model Ölçeklendirme (Kaynak: Tan ve Le, 2019).

Şekil 2.6 da (a) EfficientNet-B0 temel modelinin mimarisi. (b) Bileşik ölçeklendirme yöntemi kullanılarak modelin derinlik, genişlik ve çözünürlük boyutlarının dengeli bir şekilde nasıl ölçeklendirildiği gösterilmektedir. Bu yöntem, modelin doğruluk ve verimlilik arasında optimum bir denge kurmasını sağlar.

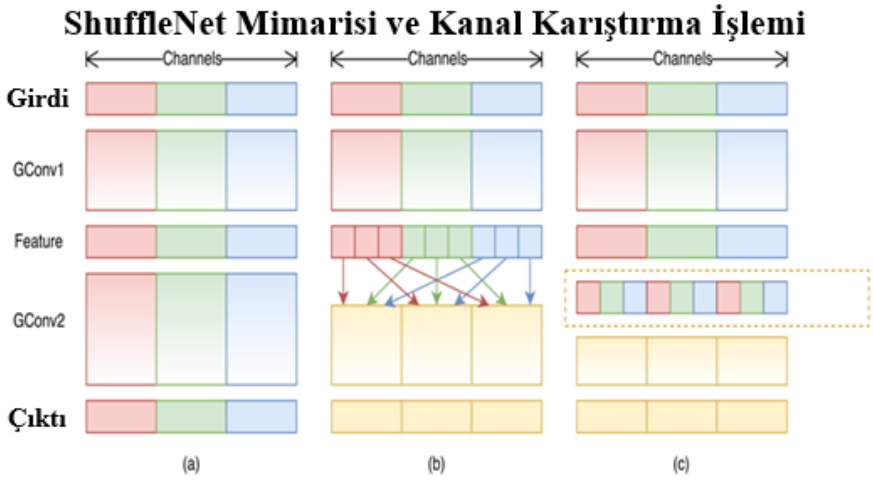
EfficientNet-B1, B2, B3, B4, B5, B6 ve B7 modelleri ise, EfficientNet-B0 modelinin farklı ölçeklendirme katsayıları kullanılarak oluşturulmuş versiyonlarıdır. Daha büyük numaralı modeller, daha fazla parametreye ve daha yüksek hesaplama maliyetine sahiptir, ancak genellikle daha yüksek doğruluk sağlarlar.

EfficientNet modelleri, yüksek doğrulukları ve görece düşük hesaplama maliyetleri ile, uç cihazlarda kullanım için iyi bir seçenek sunmaktadır.

2.5.3. ShuffleNet

ShuffleNet (Zhang ve diğerleri, 2018), gruplanmış evrişim (grouped convolution) ve kanal karıştırma (channel shuffle) işlemlerini kullanarak, hesaplama maliyetini önemli ölçüde azaltan hafif bir CNN mimarisidir. Gruplanmış evrişim, girdi kanallarını gruplara ayırarak ve her gruba ayrı ayrı evrişim uygulayarak, parametre sayısını ve hesaplama karmaşıklığını azaltır. Kanal karıştırma işlemi ise, farklı gruplardan gelen kanalları karıştırarak, gruplanmış evrişimin neden olduğu bilgi kaybını telafi eder.

ShuffleNet V1 (Zhang ve diğerleri, 2018), ShuffleNet mimarisinin ilk versiyonudur ve gruplanmış evrişim ve kanal karıştırma işlemlerini kullanarak, standart evrişim katmanlarına göre çok daha az hesaplama maliyetine sahiptir. ShuffleNet V2 (Ma ve diğerleri, 2018) ise, kanal bölme (channel split) ve birleştirme (concatenation) işlemleri gibi yeni yapısal bileşenler ekleyerek, ShuffleNet V1'in performansını ve verimliliğini daha da geliştirmiştir.



Şekil 2.7. ShuffleNet Mimarisi ve Kanal Karıştırma İşlemi. (Kaynak: Zhang ve diğerleri, 2018).

Şekil 2.7 de (a) ShuffleNet V1 için kullanılan temel yapı birimi. (b) Derinlemesine ayrılabilir evrişim (depthwise separable convolution) ve gruplanmış evrişim (grouped convolution) içeren ShuffleNet V1 yapı birimi. (c) Kanal karıştırma (channel shuffle) işlemi gösterilmektedir.

ShuffleNet modelleri, düşük hesaplama maliyetleri ve küçük boyutları ile, özellikle kaynakları kısıtlı olan uç cihazlar için uygundur.

2.6. Spesifik Uç Cihazlar: AMD NPU, Google Coral TPU, NVIDIA Jetson Nano

Bu tez çalışmasında, üç farklı uç cihaz üzerinde derin öğrenme modellerinin performans ve enerji verimliliği analiz edilmiştir: AMD NPU, Google Coral TPU ve NVIDIA Jetson Nano. Bu cihazlar, farklı donanım mimarilerine, işlemci tiplerine, bellek kapasitelerine ve enerji tüketim özelliklerine sahiptir.

2.6.1. AMD-NPU

AMD Ryzen 9 8945S işlemcisinde bulunan Ryzen AI NPU (Neural Processing Unit), yapay zeka ve derin öğrenme uygulamalarını hızlandırmak için tasarlanmış özel bir donanım birimidir. Ryzen AI NPU, XDNA mimarisi üzerine kurulmuştur ve AMD'nin ROCm platformu ve Ryzen AI Software tarafından desteklenmektedir. Bu NPU, özellikle düşük gecikme süresi ve yüksek enerji verimliliği gerektiren uygulamalar için optimize edilmiştir.

Ryzen AI NPU'nun öne çıkan özellikleri şunlardır:

Yüksek Performans: AMD Ryzen 9 8945S işlemcide, Ryzen AI NPU, yapay zeka uygulamalarında CPU ve GPU'ya kıyasla önemli ölçüde daha yüksek performans sunmaktadır.

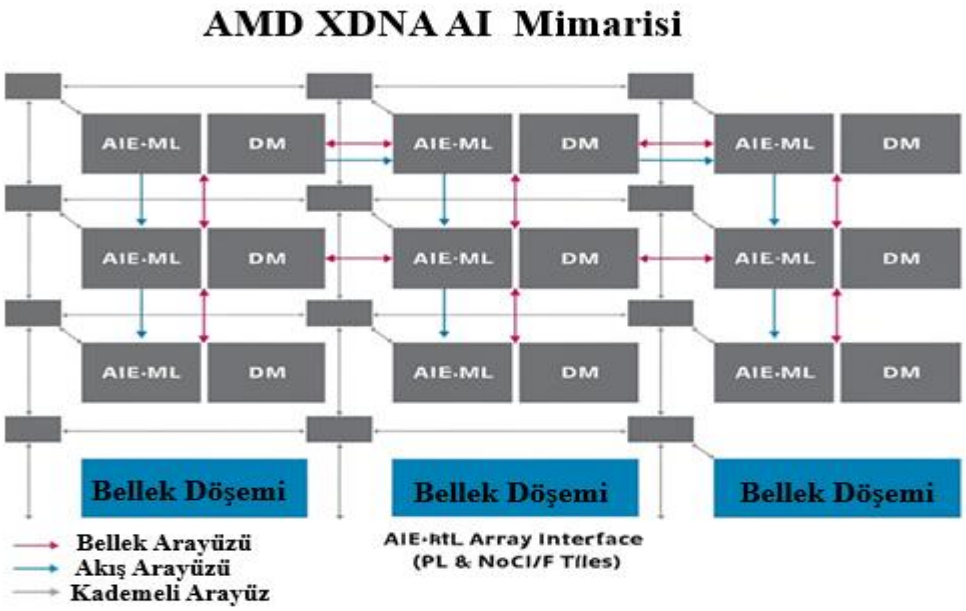
Düşük Güç Tüketimi: Ryzen AI NPU, enerji verimliliği göz önünde bulundurularak tasarlanmıştır ve düşük güç tüketimi ile yüksek performans sağlar.

Entegre Tasarım: Ryzen AI NPU, AMD Ryzen işlemci ile entegre bir şekilde çalışarak, veri aktarım gecikmelerini azaltır ve sistem performansını artırır.

Yazılım Desteği: AMD'nin ROCm platformu ve Ryzen AI Software, geliştiricilere Ryzen AI NPU'yu programlamak ve optimize etmek için gerekli araçları ve kütüphaneleri sunar.

AMD-NPU'nun mimarisi: Ryzen AI NPU, AMD'nin XDNA (Xilinx Deep Neural Network Accelerator) mimarisi üzerine kurulmuştur. XDNA, veri akışı (dataflow) mimarisine dayalı bir hızlandırıcı mimarisidir. Bu mimaride, veriler işlem birimleri arasında doğrudan aktarılır ve ara bellek erişimleri minimize edilir. Bu sayede, düşük gecikme süresi ve yüksek enerji verimliliği elde edilir. XDNA mimarisi, çok sayıda işlem birimini (PE) ve programlanabilir bir ara bağlantı yapısını (interconnect) içerir. XDNA mimarisi örneği Şekil 2.8 de gösterilmiştir. İşlem birimleri, yapay zeka uygulamalarında yaygın olarak kullanılan işlemleri (örneğin, çarpma-toplama,

aktivasyon fonksiyonları) gerçekleştirmek için optimize edilmiştir. Programlanabilir ara bağlantı yapısı ise, işlem birimleri arasında esnek ve verimli bir şekilde veri akışını sağlar.



Şekil 2.8. AMD XDNA AI Mimarisi (Kaynak: https://www.amd.com/en/technologies/xdna.html, Erişim Tarihi: 20.03.2025).

2.6.1.1. AMD-NPU'nun avantajları

Yüksek performans ve enerji verimliliği: Ryzen AI NPU, yapay zeka uygulamalarında CPU ve GPU'ya kıyasla daha yüksek performans ve enerji verimliliği sunar.

Düşük gecikme süresi: Veri akışı mimarisi sayesinde, Ryzen AI NPU, düşük gecikme süresi gerektiren uygulamalar için idealdir.

Entegrasyon: Ryzen AI NPU, AMD Ryzen işlemci ile entegre bir şekilde çalışarak, sistem performansını artırır.

Yazılım ekosistemi: AMD'nin ROCm platformu ve Ryzen AI Software, geliştiricilere kapsamlı bir yazılım ekosistemi sunar.

2.6.1.2. AMD-NPU'nun dezavantajları

Yeni teknoloji: Ryzen AI NPU, nispeten yeni bir teknolojidir ve bu nedenle, diğer hızlandırıcılar kadar geniş bir yazılım desteğine sahip olmayabilir.

Sınırlı kullanılabilirlik: Şu anda, Ryzen AI NPU, yalnızca belirli AMD Ryzen işlemci modellerinde bulunmaktadır.

TensorFlow Lite ve ONNX Runtime Desteği: Yaptığım araştırmalara göre, AMD Ryzen AI NPU, şu anda TensorFlow Lite ve ONNX Runtime kütüphanelerini destekliyor. Bu, MobileNetV3, EfficientNet-B0 ve ShuffleNet V2 modellerinin, bu formatlara dönüştürülerek NPU üzerinde çalıştırılabileceği anlamına geliyor.

ROCm ve Ryzen AI Software: Bu platformlar, AMD NPU'nun programlanmasını ve kullanılmasını kolaylaştırmakla birlikte, hala geliştirme aşamasındadır ve bazı kısıtlamalar içerebilir. Geliştiricilerin, en son güncellemeleri ve uyumluluk bilgilerini takip etmeleri önemlidir.

2.6.2. Google-Coral-TPU

Google Coral TPU (Tensor Processing Unit), Google tarafından uç cihazlarda derin öğrenme modellerinin hızlandırılması için geliştirilmiş özel bir donanım birimidir. Coral TPU, Edge TPU mimarisi üzerine kurulmuştur ve düşük gecikme süresi, yüksek enerji verimliliği ve küçük boyut gibi özellikleriyle öne çıkmaktadır. Google, Coral platformu altında, geliştiricilere farklı form faktörlerinde (örneğin, USB hızlandırıcı, PCIe hızlandırıcı, sistem modülü) ve farklı performans seviyelerinde Edge TPU çözümleri sunmaktadır.

Google Coral TPU'nun öne çıkan özellikleri:

Yüksek Performans: Edge TPU, derin öğrenme modellerinin, özellikle de Nicelenmiş (quantized) modellerin, uç cihazlarda hızlı bir şekilde çalıştırılmasını sağlar.

Düşük Güç Tüketimi: Edge TPU, enerji verimliliği göz önünde bulundurularak tasarlanmıştır ve düşük güç tüketimi ile yüksek performans sunar.

Küçük Boyut: Edge TPU, küçük boyutu sayesinde, mobil cihazlar, IoT cihazları ve gömülü sistemler gibi alan kısıtlaması olan cihazlara kolayca entegre edilebilir.

TensorFlow Lite Desteği: Edge TPU, Google'ın derin öğrenme modellerini optimize etmek ve uç cihazlarda çalıştırmak için geliştirdiği TensorFlow Lite kütüphanesini destekler.

Geliştirici Dostu: Google, Coral platformu için kapsamlı dokümantasyon, yazılım geliştirme kitleri (SDK'lar) ve örnek kodlar sağlayarak, geliştiricilerin Edge TPU'yu kolayca kullanmalarını sağlar.

Google Coral TPU'nun mimarisi:

Edge TPU, matris çarpımı işlemlerini optimize etmek için tasarlanmış özel bir donanım mimarisine sahiptir. Bu mimaride, çok sayıda çarpma-toplama (multiply-accumulate - MAC) birimi ve geniş bir on-chip bellek bulunur. Matris çarpımı, derin öğrenme modellerinin temel işlemidir ve Edge TPU, bu işlemi çok hızlı ve verimli bir şekilde gerçekleştirebilir. Ayrıca, Edge TPU, nicelenmiş modelleri donanım seviyesinde destekleyerek hem performans hem de enerji verimliliği açısından önemli avantajlar sağlar.

Şekil 2.9. Google Coral TPU (Kaynak: https://coral.ai/products/#production-products, Erişim Tarihi: 20.03.2025).

Google Coral TPU mimarisi. Edge TPU, matris çarpımı işlemlerini optimize eden özel bir donanım mimarisine sahiptir.

2.6.2.1. Google-Coral-TPU'nun avantajları

Yüksek performans ve enerji verimliliği: Edge TPU, derin öğrenme modellerini uç cihazlarda hızlı ve enerji verimli bir şekilde çalıştırır.

Düşük gecikme süresi: Edge TPU, düşük gecikme süresi gerektiren gerçek zamanlı uygulamalar için idealdir.

Küçük boyut: Edge TPU, küçük boyutu sayesinde, farklı cihazlara kolayca entegre edilebilir.

TensorFlow Lite desteği: Edge TPU, TensorFlow Lite ile tam uyumlu çalışır.

Geliştirici dostu platform: Google, Coral platformu için kapsamlı dokümantasyon ve geliştirme araçları sağlar.

2.6.2.2. Google-Coral-TPU'nun dezavantajları

Sınırlı model desteği: Edge TPU, şu anda yalnızca TensorFlow Lite modellerini desteklemektedir.

Niceleme gereksinimi: Edge TPU'dan en iyi performansı elde etmek için, modellerin Nicelenmesi (quantization) gereklidir.

2.6.3. NVIDIA Jetson Nano

NVIDIA Jetson Nano, NVIDIA tarafından uç cihazlarda yapay zeka ve derin öğrenme uygulamalarını hızlandırmak için geliştirilmiş bir geliştirme platformudur. Jetson Nano, küçük boyutu, düşük güç tüketimi ve yüksek performansı ile öne çıkmaktadır. Geliştiricilere, görüntü işleme, nesne tanıma, robotik ve otonom sistemler gibi alanlarda uygulamalar geliştirmek için güçlü ve esnek bir platform sunar.

NVIDIA Jetson Nano'nun öne çıkan özellikleri:

Yüksek Performans: Jetson Nano, 128 çekirdekli NVIDIA Maxwell GPU'su ve dört çekirdekli ARM Cortex-A57 CPU'su sayesinde, derin öğrenme modellerini uç cihazlarda yüksek performansla çalıştırabilir.

Düşük Güç Tüketimi: Jetson Nano, enerji verimliliği göz önünde bulundurularak tasarlanmıştır ve 5W ile 10W arasında bir güç tüketimi ile çalışır.

Küçük Boyut: Jetson Nano, 70mm x 45mm boyutlarındadır ve bu sayede, alan kısıtlaması olan cihazlara kolayca entegre edilebilir.

Geniş Yazılım Desteği: Jetson Nano, NVIDIA JetPack SDK tarafından desteklenmektedir. JetPack SDK, CUDA, cuDNN, TensorRT gibi derin öğrenme kütüphanelerini ve araçlarını içerir. Ayrıca, Linux tabanlı bir işletim sistemi ile birlikte gelir.

Geliştirici Dostu: NVIDIA, Jetson Nano için kapsamlı dokümantasyon, eğitim materyalleri ve örnek kodlar sağlayarak, geliştiricilerin platformu kolayca kullanmalarını sağlar.

NVIDIA Jetson Nano'nun mimarisi:

Jetson Nano, NVIDIA Maxwell GPU mimarisi üzerine kurulmuş 128 çekirdekli bir GPU ve dört çekirdekli ARM Cortex-A57 CPU içermektedir. GPU, derin öğrenme modellerinin paralel işlenmesini hızlandırmak için CUDA (Compute Unified Device Architecture) platformunu kullanır. CPU ise, genel amaçlı işlemler ve sistem yönetimi için kullanılır. Jetson Nano, ayrıca 4 GB LPDDR4 bellek, microSD kart yuvası, USB 3.0 ve 2.0 portları, HDMI ve DisplayPort görüntü çıkışları, Gigabit Ethernet portu ve MIPI CSI kamera arayüzü gibi zengin bağlantı seçeneklerine sahiptir.

Şekil 2.10. NVIDIA Jetson Nano (Kaynak: https://www.nvidia.com/tr-tr/autonomous-machines/embedded-systems/jetson-nano/product-development, Erişim Tarihi: 20.03.2025).

2.6.3.1. NVIDIA Jetson Nano'nun avantajları

Yüksek performans: Jetson Nano, GPU hızlandırması sayesinde, derin öğrenme modellerini uç cihazlarda yüksek performansla çalıştırır. Düşük güç tüketimi: Jetson Nano, enerji verimliliği sunar ve düşük güç tüketimi ile çalışır.

Küçük boyut: Jetson Nano, kompakt boyutu sayesinde, farklı cihazlara kolayca entegre edilebilir.

Geniş yazılım desteği: Jetson Nano, NVIDIA'nın kapsamlı yazılım ekosistemi tarafından desteklenmektedir.

Geliştirici dostu platform: NVIDIA, Jetson Nano için kapsamlı dokümantasyon ve geliştirme araçları sağlar.

2.6.3.2. NVIDIA Jetson Nano'nun dezavantajları

Fiyat: Jetson Nano, diğer uç cihazlara (örneğin, Raspberry Pi) kıyasla daha pahalıdır.

GPU kullanımı için yazılım optimizasyonu gerekliliği: Jetson Nano'nun GPU'sundan tam olarak yararlanmak için, modellerin ve uygulamaların CUDA ve TensorRT gibi kütüphaneler kullanılarak optimize edilmesi gerekir.

TensorFlow Lite ile GPU kullanımının sınırlı olması: TensorFlow Lite kütüphanesinin Python tarafında, Jetson Nano'nun GPU'su için doğrudan bir destek sunmaması, bu çalışmada GPU kullanımını engellemiştir.

3. MATERYAL VE YÖNTEM

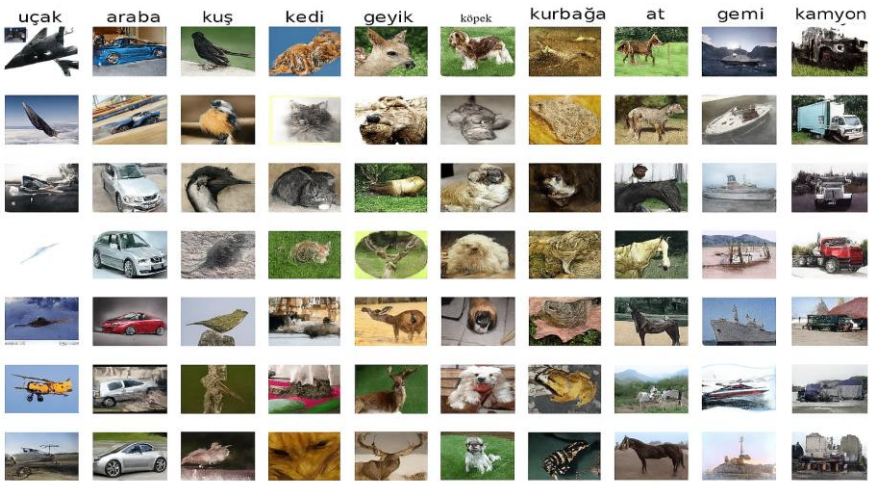
Bu bölümde, tez kapsamında yapılan çalışmada kullanılan veri setleri, derin öğrenme modelleri, model sıkıştırma teknikleri, uygulama ortamı ve cihazlar, değerlendirme metrikleri ve deney tasarımı detaylı bir şekilde açıklanmaktadır.

3.1. Veri Setleri

Bu çalışmada, üç farklı görüntü sınıflandırma veri seti kullanılmıştır: CIFAR-10, CIFAR-100 ve ImageNet (alt küme). Bu veri setleri, derin öğrenme modellerinin performansını değerlendirmek için yaygın olarak kullanılan standart veri setleridir.

3.1.1. CIFAR-10

Canadian Institute for Advanced Research CIFAR-10 veri seti, 10 farklı nesne sınıfına ait 60.000 adet 32x32 piksel boyutunda renkli görüntülerden oluşmaktadır. Her sınıf, 6.000 görüntü içermektedir. Veri seti, 50.000 eğitim görüntüsü ve 10.000 test görüntüsü olmak üzere ikiye ayrılmıştır. CIFAR-10 veri setinde bulunan sınıflar şunlardır: uçak, otomobil, kuş, kedi, geyik, köpek, kurbağa, at, gemi ve kamyon (Kaynak: Krizhevsky, A. (2009). CIFAR-10 and CIFAR-100 datasets.). Eğitimde kullanılan görsellerin sınıf dağılımı ve çeşitliliği, Şekil 3.1'de gösterilen CIFAR-10 veri seti örneklerinde açıkça görülmektedir.



Şekil 3.1. CIFAR-10 Veri Setinden Örnek Görüntüler (Kaynak: Krizhevsky, 2009).

3.1.2. CIFAR-100

Canadian Institute for Advanced Research CIFAR-100 veri seti, CIFAR-10'a benzer şekilde, 100 farklı nesne sınıfına ait 60.000 adet 32x32 piksel boyutunda renkli görüntülerden oluşmaktadır. Her sınıf, 600 görüntü içermektedir. Veri seti, 50.000 eğitim görüntüsü ve 10.000 test görüntüsü olmak üzere ikiye ayrılmıştır. CIFAR-100 veri setindeki sınıflar, tematik olarak gruplandırılmış ve her grup 20 alt sınıfa ayrılmıştır. (Örneğin; balıklar, çiçekler, ev aletleri, böcekler vs.) Şekil 3.2'de gösterilen CIFAR-100 veri seti örneklerinde açıkça görülmektedir.

Şekil 3.2. CIFAR-100 Veri Setinden Örnek Görüntüler (Kaynak: Krizhevsky, 2009).

3.1.3. ImageNet (Alt küme)

ImageNet veri seti, 1000'den fazla nesne sınıfına ait 1.2 milyondan fazla yüksek çözünürlüklü renkli görüntüden oluşan, geniş kapsamlı bir veri setidir. Bu çalışmada, ImageNet veri setinin tamamı yerine, 10 farklı sınıftan (kedi, köpek, araba, uçak, kuş, gemi, çiçek, meyve, insan, manzara) rastgele seçilmiş 1000'er görüntü içeren bir alt küme kullanılmıştır. Bu alt küme, toplamda 10.000 görüntüden (9.000 eğitim, 1.000 test) oluşmaktadır. Şekil3.3'te ImageNet veri setinden örnek görüntüler görülmektedir.

Şekil 3.3. ImageNet Veri Setinden Örnek Görüntüler (Kaynak: Deng, 2009).

Veri Seti Seçim Kriterleri: Bu çalışmada, CIFAR-10, CIFAR-100 ve ImageNet veri setlerinin seçilmesinin temel nedenleri şunlardır:

Yaygın Kullanım: Bu veri setleri, görüntü sınıflandırma alanında yaygın olarak kullanılan ve model performanslarının karşılaştırılması için bir standart oluşturan veri setleridir.

Çeşitlilik: Farklı sayıda sınıfa, görüntü boyutuna ve nesne türüne sahip olmaları, modellerin genelleme yeteneğinin farklı açılardan değerlendirilmesini sağlar.

Erişilebilirlik: Bu veri setleri, açık kaynaklı olarak erişilebilir durumdadır ve araştırmacılar tarafından kolayca kullanılabilir.

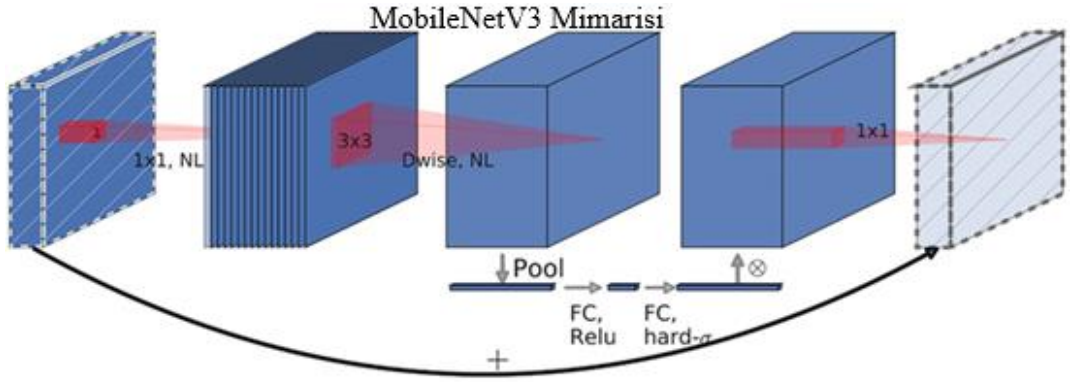
ImageNet Alt Kümesinin Seçilme Nedeni: ImageNet veri setinin tamamının kullanılması, eğitim ve test süreçlerinin çok uzun sürmesine ve yüksek hesaplama kaynakları gerektirmesine neden olacaktı. Bu nedenle, daha hızlı ve verimli bir şekilde deney yapabilmek için, 10 farklı sınıftan 1000'er görüntü içeren bir alt küme kullanılmıştır.

3.2. Model Mimarileri

Bu çalışmada, uç cihazlarda verimli bir şekilde çalışabilen hafif (lightweight) derin öğrenme modelleri tercih edilmiştir. Bu kapsamda, MobileNetV3, EfficientNet-B0 ve ShuffleNet V2 olmak üzere üç farklı model kullanılmıştır. Bu modeller, düşük hesaplama karmaşıklığı, az sayıda parametre ve düşük bellek kullanımı gibi özellikleriyle öne çıkmaktadır.

3.2.1. MobileNetV3

MobileNetV3 (Howard ve diğerleri, 2019), derinlemesine ayrılabilir evrişim (depthwise separable convolution) ve ters çevrilmiş artık bloklar (inverted residual blocks) yapılarını kullanarak, model boyutunu ve hesaplama maliyetini önemli ölçüde azaltan bir mimaridir. Ayrıca, ağ mimarisi arama (NAS) yöntemleriyle optimize edilmiş ve h-swish aktivasyon fonksiyonu gibi platforma özgü iyileştirmeler içermektedir.



Şekil 3.4. MobileNetV3 mimarisi. (Kaynak: Howard ve diğerleri, 2019).

Derinlemesine ayrılabilir evrişim, ters çevrilmiş artık bloklar ve ağ mimarisi arama (NAS) optimizasyonları, modelin hafif ve verimli olmasını sağlamaktadır. MobileNetV3 mimarisi Şekil 3.4 de gösterilmiştir.

Tercih Edilme Nedenleri: MobileNetV3, yüksek doğruluk oranı, düşük çıkarım süresi ve düşük bellek kullanımı ile uç cihazlar için optimize edilmiş bir modeldir. Ayrıca, TensorFlow Lite ve Edge TPU gibi platformlarla uyumludur.

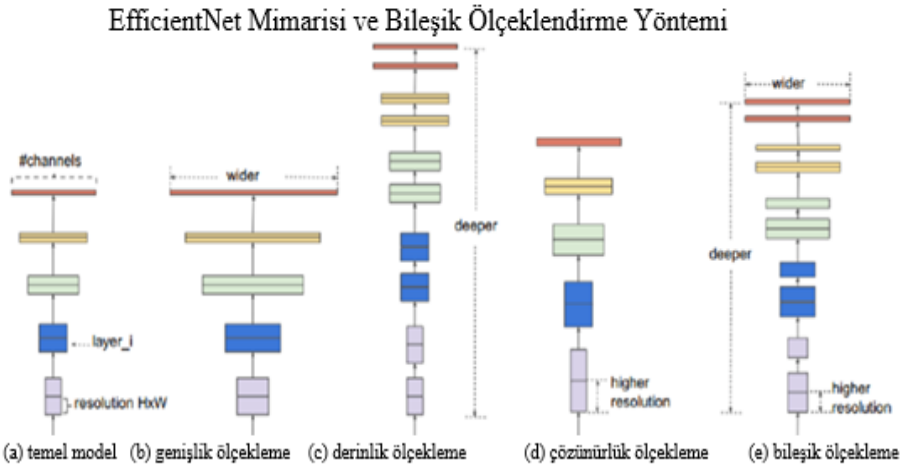
Avantajları: Hızlı çıkarım süresi, Düşük enerji tüketimi, Küçük model boyutu, Yüksek doğruluk oranı.

Dezavantajları: Çok büyük ve karmaşık veri setlerinde, daha büyük ve derin modeller kadar yüksek doğruluk sağlayamayabilir. Mimari arama (NAS) ile optimize edildiği için, mimarinin anlaşılması ve özelleştirilmesi diğer modellere göre daha zor olabilir.

3.2.2. EfficientNet-B0

EfficientNet-B0 (Tan ve Le, 2019), bileşik ölçeklendirme (compound scaling) adı verilen bir yöntem kullanarak, modelin derinliğini, genişliğini ve çözünürlüğünü dengeli bir şekilde ölçeklendiren bir CNN mimarisidir. Bileşik ölçeklendirme, modelin

tüm boyutlarının (derinlik, genişlik, çözünürlük) belirli bir katsayı ile çarpılarak büyütülmesi veya küçültülmesi işlemidir. Bu yöntem, modelin performansını ve verimliliğini optimize etmek için, her bir boyutun nasıl ölçeklendirileceğini belirler.



Şekil 3.5. EfficientNet Mimarisi ve Bileşik Ölçeklendirme Yöntemi (Kaynak: Tan ve Le, 2019).

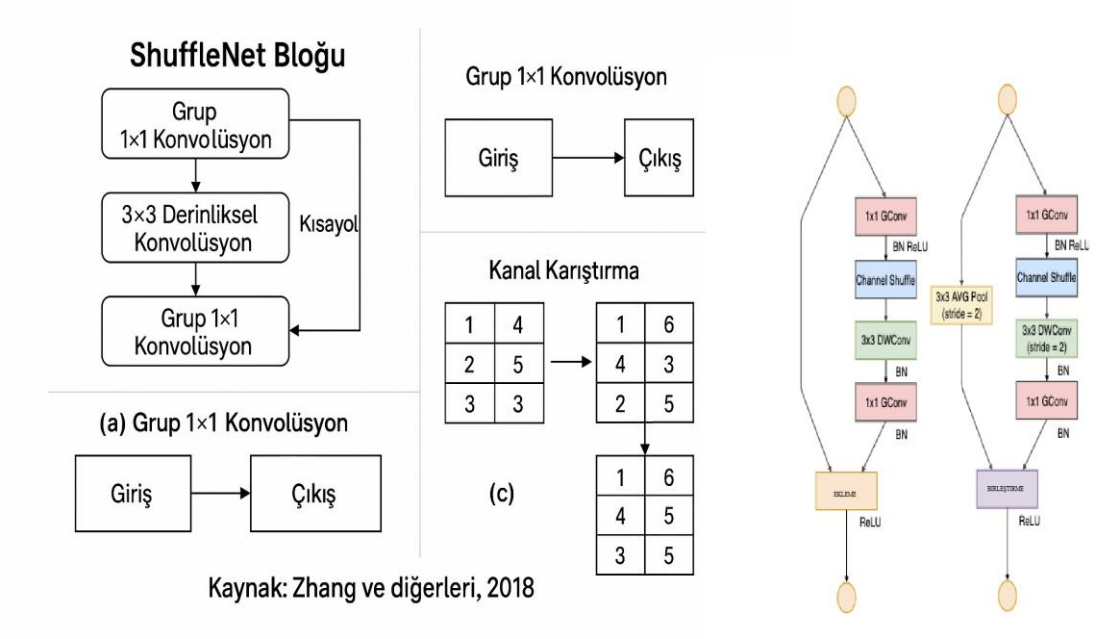
Tercih Edilme Nedenleri: EfficientNet-B0, yüksek doğruluk oranı ve görece düşük hesaplama maliyeti sunan bir modeldir. Şekil 3.5'de gösterilen bileşik ölçeklendirme yöntemi, modelin farklı donanım platformlarına uyarlanmasını kolaylaştırır. Modelin derinliği, genişliği ve çözünürlüğü, dengeli bir şekilde ölçeklendirilerek, yüksek doğruluk ve verimlilik elde edilmektedir.

Avantajları: Yüksek doğruluk oranı, İyi bir verimlilik ve doğruluk dengesi, Ölçeklenebilir bir mimariye sahip olması.

Dezavantajları: MobileNet ve ShuffleNet'e kıyasla daha yüksek hesaplama maliyetine ve model boyutuna sahip olabilir. Bileşik ölçeklendirme yöntemi, modelin yorumlana bilirliğini ve anlaşılırlığını zorlaştırabilir.

3.2.3. ShuffleNetV2

ShuffleNet V2 (Ma ve diğerleri, 2018), gruplanmış evrişim (grouped convolution) ve kanal karıştırma (channel shuffle) tekniklerini kullanarak, hesaplama maliyetini önemli ölçüde azaltan bir mimaridir. Ayrıca, kanal bölme (channel split) ve birleştirme (concatenation) işlemleri ile daha da optimize edilmiştir.



Şekil 3.6. ShuffleNet Mimarisi ve Kanal Karıştırma İşlemi (Kaynak: Zhang ve diğerleri, 2018).

Gruplanmış evrişim ve kanal karıştırma, modelin hesaplama maliyetini önemli ölçüde azaltmaktadır bu durum Şekil 3.6 da gösterilmektedir. Tercih Edilme Nedenleri: ShuffleNet V2, çok düşük hesaplama maliyeti ve küçük model boyutu ile öne çıkmaktadır. Bu özellikleri, modeli özellikle kaynakları kısıtlı olan uç cihazlar için uygun hale getirmektedir.

Avantajları: Çok düşük hesaplama maliyeti, çok küçük model boyutu, yüksek hız.

Dezavantajları: Diğer hafif modeller kadar yüksek doğruluk sağlamayabilir. Çok fazla kanal karıştırma işlemi, modelin eğitimini zorlaştırabilir.

3.3. Model Sıkıştırma Teknikleri

Bu çalışmada, derin öğrenme modellerinin boyutunu ve hesaplama karmaşıklığını azaltmak için Budama (pruning) ve Niceleme (quantization) olmak üzere iki farklı model sıkıştırma tekniği kullanılmıştır.

3.3.1. Budama (Pruning)

Budama, modeldeki gereksiz bağlantıların, nöronların veya filtrelerin kaldırılması işlemidir. Bu sayede, modelin boyutu küçülür, hesaplama karmaşıklığı azalır ve enerji verimliliği artar. Bu çalışmada, ağırlık Budama (weight pruning) ve filtre Budama (filter pruning) teknikleri uygulanmıştır.

3.3.1.1. Ağırlık budama (Weight pruning)

Yöntem: Büyüklüğe dayalı Budama (magnitude-based pruning) yöntemi kullanılmıştır. Bu yöntemde, eğitilmiş modeldeki ağırlıkların mutlak değerleri belirli bir eşik değerinin altında olanlar sıfırlanarak model seyreltilir.

Uygulama: Ağırlık Budama, TensorFlow Model Optimization Toolkit kullanılarak gerçekleştirilmiştir. Budama işlemi, modellerin eğitiminden sonra, "tfmot.sparsity.keras.prune_low_magnitude" fonksiyonu ile uygulanmıştır. Bu fonksiyon, belirtilen hedef seyreklik oranına ulaşana kadar, modeli yinelemeli olarak budar.

Budama Oranları: Farklı Budama oranlarının etkisini incelemek için, %20, %40, %60 ve %80 olmak üzere dört farklı Budama oranı uygulanmıştır. Örneğin, %20 Budama oranı, modeldeki ağırlıkların en küçük %20'sinin sıfırlanacağı anlamına gelmektedir.

Katman Seçimi: Ağırlık Budama işlemi, modellerin tüm katmanlarına uygulanmıştır.

İterasyon Sayısı: Ağırlık Budama işlemi, 10 yineleme (iteration) boyunca gerçekleştirilmiştir. Her yinelemede, belirli bir oranda ağırlık sıfırlanmış ve model yeniden eğitilmiştir (fine-tuning).

Fine-tuning: Budama işleminden sonra, Budamadan kaynaklı olası doğruluk kayıplarını telafi etmek için, modeller düşük bir öğrenme oranıyla (0.0001) ve 5 epoch boyunca tekrar eğitilmiştir.

3.3.1.2. Filtre Budama (Filter Pruning)

Yöntem: L1 normuna dayalı filtre Budama yöntemi kullanılmıştır. Bu yöntemde, her bir filtrenin L1 normu (ağırlıkların mutlak değerlerinin toplamı) hesaplanır ve en düşük L1 normuna sahip filtreler, modelden kaldırılır.

Uygulama: Filtre Budama, TensorFlow Model Optimization Toolkit kullanılarak gerçekleştirilmiştir. Budama işlemi, modellerin eğitiminden sonra, "tfmot.sparsity.keras.prune_low_magnitude" fonksiyonu ve "tfmot.sparsity.keras.PruningPolicy" arayüzü ile uygulanmıştır. "L1NormPruningPolicy" kullanılarak, filtrelerin L1 normuna göre budanması sağlanmıştır.

Budama Oranları: Farklı Budama oranlarının etkisini incelemek için, %20, %40, %60 ve %80 olmak üzere dört farklı Budama oranı uygulanmıştır.

Katman Seçimi: Filtre Budama işlemi, evrişimli katmanlardaki (convolutional layers) filtrelere uygulanmıştır.

İterasyon Sayısı: Filtre Budama işlemi, 10 yineleme (iteration) boyunca gerçekleştirilmiştir. Her yinelemede, belirli bir oranda filtre kaldırılmış ve model yeniden eğitilmiştir (fine-tuning).

Fine-Tuning: Budama işleminden sonra, modeller düşük bir öğrenme oranıyla (0.0001) ve 5 epoch boyunca tekrar eğitilmiştir.

3.3.2. Niceleme (Quantization)

Niceleme, modelin ağırlıklarının ve aktivasyonlarının daha düşük bit derinliğine (örneğin, 32-bit float'tan 8-bit integer'a) dönüştürülmesi işlemidir. Bu sayede, modelin boyutu küçülür, bellek kullanımı azalır ve hesaplama hızı artar. Bu çalışmada, eğitim sonrası niceleme (post-training quantization - PTQ) ve niceleme farkındalıklı eğitim (quantization-aware training - QAT) teknikleri uygulanmıştır.

3.3.2.1. Eğitim sonrası niceleme (Post training quantization (PTQ))

Yöntem: Tam sayı niceleme (integer-only quantization) yöntemi kullanılmıştır. Bu yöntemde, eğitilmiş modelin ağırlıkları ve aktivasyonları, 8-bit tamsayılara dönüştürülür.

Uygulama: PTQ, TensorFlow Lite Converter kullanılarak gerçekleştirilmiştir. Model, "tf.lite.TFLiteConverter.from_keras_model" fonksiyonu ile TensorFlow Lite formatına dönüştürülmüş ve "converter.optimizations = [tf.lite.Optimize.DEFAULT]" seçeneği ile tam sayı Niceleme uygulanmıştır.

Kalibrasyon Veri Seti: PTQ işlemi sırasında, Niceleme parametrelerini (ölçek ve ofset değerleri) belirlemek için, eğitim veri setinden rastgele seçilmiş 1000 görüntüden oluşan bir kalibrasyon veri seti kullanılmıştır.

Dinamik Aralık Niceleme: Google Coral TPU'da etkin bir Niceleme için dinamik aralık Niceleme (dynamic range quantization) yöntemi kullanılmıştır.

3.3.2.2. Niceleme farkındalıklı eğitim (Quantization aware training (QAT))

Yöntem: Eğitim sırasında, Nicelemenin etkileri simüle edilerek, model Nicelenmiş ağırlıklara ve aktivasyonlara göre eğitilir. Bu sayede, nicelemeden kaynaklı doğruluk kaybı minimize edilir.

Uygulama: QAT, TensorFlow Model Optimization Toolkit kullanılarak gerçekleştirilmiştir. Model, "tfmot.quantization.keras.quantize_model" fonksiyonu ile niceleme farkındalıklı hale getirilmiş ve ardından tekrar eğitilmiştir.

Eğitim Süreci: QAT işlemi, 10 epok boyunca gerçekleştirilmiştir. Her epokta, model nicelenmiş ağırlıklar ve aktivasyonlar kullanılarak eğitilmiş ve doğruluk değeri izlenmiştir.

3.4. Uygulama Ortamı ve Cihazlar

Bu tez çalışmasında, deneyler üç farklı uç cihaz üzerinde gerçekleştirilmiştir: AMD NPU, Google Coral TPU ve NVIDIA Jetson Nano. Bu cihazlar, farklı donanım mimarilerine, işlemci tiplerine, bellek kapasitelerine ve enerji tüketim özelliklerine sahiptir.

3.4.1. AMD-NPU

AMD Ryzen 9 8945S işlemcili bir dizüstü bilgisayar kullanılmıştır. Bu işlemci, yapay zeka ve derin öğrenme uygulamalarını hızlandırmak için entegre bir Ryzen AI NPU (Neural Processing Unit) içermektedir. NPU, XDNA mimarisi üzerine kurulmuştur ve AMD'nin ROCm platformu ve Ryzen AI Software tarafından desteklenmektedir.

Teknik Özellikler:

- İşlemci: AMD Ryzen 9 8945S
- NPU: Ryzen AI (XDNA mimarisi)
- Çekirdek Sayısı: 8 Çekirdek, 16 İş Parçacığı
- Frekans: 3.6 GHz (Temel), 4.9 GHz (Turbo)
- Bellek: 32 GB DDR5 5600 MHz
- İşletim Sistemi: Windows 11

Yazılım Ortamı:

- Ryzen AI Software: 1.0 (Sürücü ve kütüphaneler)
- ROCm Platformu: 5.7.1
- ONNX Runtime: 1.13.1 (NPU üzerinde çalıştırmak için gerekli kütüphaneler ve araçlar, AMD'nin sağladığı dokümantasyona göre kurulmuştur.)
- TensorFlow: 2.9.1 (Model eğitimi için)
- TensorFlow Model Optimization Toolkit: 0.7.3 (Model sıkıştırma için)

- tf2onnx: 1.14.0 (Modeli ONNX formatına dönüştürmek için)
- NumPy: 1.23.4
- Pandas: 1.5.1
- Scikit-learn: 1.2.0
- Matplotlib: 3.6.2
- psutil: 5.9.5

Kurulum:

Ryzen AI Software ve ROCm Platformu: AMD'nin resmi web sitesinden indirilen kurulum dosyaları kullanılarak kurulmuştur.

ONNX Runtime: pip install onnxruntime-dml komutu ile kurulmuştur.

TensorFlow: pip install tensorflow==2.9.1 komutu ile kurulmuştur.

TensorFlow Model Optimization Toolkit: pip install tensorflow-model-optimization==0.7.3 komutu ile kurulmuştur.

Diğer Kütüphaneler: pip install numpy pandas scikit-learn matplotlib psutil komutu ile kurulmuştur.

Gerekli Sürücüler: AMD'nin sağladığı en güncel sürücüler kurulmuştur.

Ortam Değişkenleri: Gerekli ortam değişkenleri (örneğin, PATH) AMD'nin sağladığı dokümantasyona göre ayarlanmıştır.

NPU'nun Kullanımı: Deneylerde, TensorFlow ile eğitilmiş ve Keras formatında kaydedilmiş modeller, öncelikle ONNX formatına dönüştürülmüş, ardından ONNX Runtime aracılığıyla Ryzen AI NPU üzerinde çalıştırılmıştır.

Modelin ONNX Formatına Dönüştürülmesi:

TensorFlow/Keras modeli ONNX formatına dönüştürmek için "tf2onnx" kütüphanesi kullanılmıştır.

NPU Üzerinde Çıkarım Yapma:

ONNX Runtime, "DmlExecutionProvider" kullanılarak, modelin AMD NPU üzerinde çalıştırılması sağlanmıştır.

AMD NPU'nun Seçilme Nedenleri: AMD Ryzen AI NPU, yapay zeka uygulamalarında yüksek performans ve enerji verimliliği sunan yeni bir teknolojidir.

Bu çalışmada, NPU'nun derin öğrenme modellerinin uç cihazlarda çalıştırılması için potansiyelini değerlendirmek amacıyla seçilmiştir.

AMD NPU'nun Avantajları:

Yüksek Performans: Ryzen AI NPU, yapay zeka uygulamalarında CPU ve GPU'ya kıyasla daha yüksek performans sunar.

Enerji Verimliliği: Düşük güç tüketimi ile yüksek performans sağlar.

Entegrasyon: Ryzen AI NPU, AMD Ryzen işlemci ile entegre bir şekilde çalışarak, sistem performansını artırır.

Yazılım Ekosistemi: AMD'nin ROCm platformu ve Ryzen AI Software, geliştiricilere kapsamlı bir yazılım ekosistemi sunar.

ONNX Runtime Desteği: AMD NPU, ONNX Runtime modellerini destekler.

AMD NPU'nun Dezavantajları:

Yeni Teknoloji: Ryzen AI NPU, nispeten yeni bir teknolojidir ve bu nedenle, diğer hızlandırıcılar kadar geniş bir yazılım desteğine sahip olmayabilir.

Sınırlı Kullanılabilirlik: Şu anda, Ryzen AI NPU, yalnızca belirli AMD Ryzen işlemci modellerinde bulunmaktadır.

ROCm ve Ryzen AI Software: Bu platformlar, AMD NPU'nun programlanmasını ve kullanılmasını kolaylaştırmakla birlikte, hala geliştirme aşamasındadır ve bazı kısıtlamalar içerebilir. Geliştiricilerin, en son güncellemeleri ve uyumluluk bilgilerini takip etmeleri önemlidir.

3.4.2. GoogleCoral-TPU

Google Coral USB Accelerator ve Google Coral Dev Board Mini cihazları kullanılmıştır. Bu cihazlar, derin öğrenme modellerinin düşük gecikme süresi ve yüksek enerji verimliliği ile çalıştırılmasını sağlayan Edge TPU (Tensor Processing Unit) içermektedir.

Teknik Özellikler:

İşlemci: Edge TPU (Google Coral USB Accelerator ve Dev Board Mini'de aynı)

Frekans: 4 TOPS (Trilyon İşlem/Saniye)

Bellek: (Google Coral USB Accelerator: Harici bellek kullanmaz, Google Coral Dev Board Mini: 2 GB LPDDR4)

İşletim Sistemi: Mendel Linux (Google Coral Dev Board Mini için)

Yazılım Ortamı:

Edge TPU Runtime: 14.1

TensorFlow Lite: 2.9.1

Python: 3.10

NumPy: 1.23.4

Kurulum:

Edge TPU Runtime: Google Coral'ın resmi web sitesinden indirilen kurulum dosyaları kullanılarak kurulmuştur.

TensorFlow Lite: pip install tflite-runtime komutu ile kurulmuştur.

Diğer Kütüphaneler: İhtiyaç duyulan diğer kütüphaneler (örneğin, NumPy), pip install <kütüphane_adı> komutu ile kurulmuştur.

TPU'nun Kullanımı: Deneylerde, TensorFlow Lite modelleri, Edge TPU Runtime kütüphanesi kullanılarak Google Coral TPU üzerinde çalıştırılmıştır. TensorFlow Lite modelleri, "tflite_runtime.interpreter.Interpreter" sınıfı ile yüklenmiş ve "allocate_tensors()", "set_tensor()", "invoke()" ve "get_tensor()" fonksiyonları kullanılarak çıkarım yapılmıştır.

Google Coral TPU'nun Seçilme Nedenleri: Google Coral TPU, düşük güç tüketimi, yüksek enerji verimliliği ve hızlı çıkarım süresi ile uç cihazlarda derin öğrenme uygulamaları için ideal bir platformdur. Ayrıca, TensorFlow Lite ile tam uyumlu olması ve Google tarafından sağlanan kapsamlı geliştirici desteği, bu cihazın tercih edilmesinde önemli rol oynamıştır.

Google Coral TPU'nun Avantajları:

Yüksek Performans: Düşük gecikme süresi ile hızlı çıkarım sağlar.

Enerji Verimliliği: Düşük güç tüketimi ile yüksek performans sunar.

Küçük Boyut: Farklı form faktörlerinde (USB, PCIe, SOM) mevcut olması, çeşitli cihazlara entegrasyonunu kolaylaştırır.

TensorFlow Lite Desteği: TensorFlow Lite ile tam uyumlu çalışır.

Geliştirici Desteği: Google, kapsamlı dokümantasyon, SDK'lar ve örnek kodlar sağlar.

Google Coral TPU'nun Dezavantajları:

Sınırlı Model Desteği: Sadece TensorFlow Lite modellerini destekler.

Niceleme Gereksinimi: En iyi performansı elde etmek için modellerin Nicelenmesi gerekir.

3.4.3. NVIDIA jetson nano

NVIDIA jetson nano geliştirici kiti kullanılmıştır. Bu kit, 128 çekirdekli NVIDIA Maxwell GPU'su ve dört çekirdekli ARM Cortex-A57 CPU'su ile derin öğrenme uygulamaları için yüksek performans sunmaktadır.

Teknik Özellikler:

İşlemci: Dört Çekirdekli ARM Cortex-A57

GPU: 128 çekirdekli NVIDIA Maxwell

Frekans: 1.43 GHz (CPU), 921 MHz (GPU)

Bellek: 4 GB LPDDR4

İşletim Sistemi: Linux for Tegra (L4T)

Yazılım Ortamı:

NVIDIA JetPack SDK: 4.6

CUDA: 10.2

cuDNN: 8.2.1

TensorRT: 8.0.1.6

TensorFlow: 2.9.1 (Jetson Nano'da GPU desteği ile kullanmak için NVIDIA'nın sağladığı özel TensorFlow paketleri kullanılmalıdır).

PyTorch: 1.10.0 (Jetson Nano'da GPU desteği ile kullanmak için NVIDIA'nın sağladığı özel PyTorch paketleri kullanılmalıdır).

Kurulum:

JetPack SDK: NVIDIA'nın resmi web sitesinden indirilen JetPack SDK kullanılarak Jetson Nano'ya kurulmuştur.

TensorFlow ve PyTorch: NVIDIA'nın sağladığı özel paketler ve kurulum talimatları kullanılarak, GPU desteği ile kurulmuştur.

Gerekli Sürücüler: NVIDIA'nın sağladığı en güncel sürücüler kurulmuştur.

GPU'nun Kullanımı: Bu çalışmada, Jetson Nano'nun GPU'sundan yararlanmak hedeflenmişti. Ancak, TensorFlow Lite kütüphanesinin Python tarafında, Jetson Nano'nun GPU'su için doğrudan bir destek sunmadığı fark edilmiştir. Bu nedenle, deneylerde TensorFlow Lite modelleri, Jetson Nano'nun CPU'su üzerinde çalıştırılmıştır. TensorRT kütüphanesi, Jetson Nano'nun GPU'sunda derin öğrenme modellerini çalıştırmak için kullanılabilir, ancak bu çalışma kapsamında sadece TensorFlow Lite modelleri ve CPU kullanımı ele alınmıştır.

NVIDIA Jetson Nano'nun Seçilme Nedenleri: Jetson Nano, yüksek performansı, düşük güç tüketimi ve geniş yazılım desteği ile uç cihazlarda derin öğrenme uygulamaları için güçlü bir platformdur. Ayrıca, NVIDIA'nın sağladığı kapsamlı dokümantasyon ve geliştirici araçları, bu cihazın kullanımını kolaylaştırmaktadır.

NVIDIA Jetson Nano'nun Avantajları:

Yüksek Performans: GPU hızlandırması sayesinde, derin öğrenme modellerini uç cihazlarda yüksek performansla çalıştırır.

Düşük Güç Tüketimi: Enerji verimliliği sunar ve düşük güç tüketimi ile çalışır.

Küçük Boyut: Kompakt boyutu sayesinde, farklı cihazlara kolayca entegre edilebilir.

Geniş Yazılım Desteği: NVIDIA, kapsamlı bir yazılım ekosistemi ve geliştirici araçları sağlar.

Geliştirici Dostu Platform: NVIDIA, Jetson Nano için kapsamlı dokümantasyon ve geliştirme araçları sağlar.

NVIDIA Jetson Nano'nun Dezavantajları:

Fiyat: Jetson Nano, diğer uç cihazlara (örneğin, Raspberry Pi) kıyasla daha pahalıdır.

GPU Kullanımı İçin Yazılım Optimizasyonu Gerekliliği: Jetson Nano'nun GPU'sundan tam olarak yararlanmak için, modellerin ve uygulamaların CUDA ve TensorRT gibi kütüphaneler kullanılarak optimize edilmesi gerekir.

TensorFlow Lite ile GPU Kullanımının Sınırlı Olması: TensorFlow Lite kütüphanesinin Python tarafında, Jetson Nano'nun GPU'su için doğrudan bir destek sunmaması, bu çalışmada GPU kullanımını engellemiştir.

Deneyler sırasında, cihazların enerji tüketimini ölçmek için Intertek JGQ02S-01 enerji ölçüm cihazı kullanılmıştır.

3.5. Yazılım Ortamı

Bu tez çalışmasında, derin öğrenme modellerini eğitmek, optimize etmek ve çalıştırmak için Python 3.10 sürümü kullanılmıştır. Model eğitimi ve optimizasyonu için TensorFlow 2.9.1 ve PyTorch 1.10.0 kütüphaneleri kullanılmıştır. Model sıkıştırma işlemleri için TensorFlow Model Optimization Toolkit 0.7.3 kütüphanesinden yararlanılmıştır. Ayrıca, NumPy 1.23.4, Pandas 1.5.1, Scikit-learn 1.2.0 ve Matplotlib 3.6.2 gibi veri işleme ve görselleştirme kütüphaneleri kullanılmıştır. Uç cihazlarda modellerin çalıştırılması için gerekli kütüphaneler (örneğin, TensorFlow Lite, TensorRT) ilgili cihazların dokümantasyonlarında belirtilen şekilde kurulmuştur.

Yazılım ve kütüphane versiyonları:

Bu çalışmada gerçekleştirilen deneylerin tekrarlanabilirliği ve sonuçların doğrulanabilirliği amacıyla, kullanılan tüm yazılım ve kütüphane versiyonları titizlikle belirlenmiş; ilgili resmi dökümantasyonlar ve güvenilir kaynaklara dayandırılarak raporlanmıştır.

3.5.1. Programlama dili ve temel kütüphaneler

Python: 3.10

Çalışmanın temel programlama dili olarak Python 3.10 tercih edilmiştir. Python'un geniş kütüphane desteği, bilimsel ve mühendislik uygulamalarında yaygın olarak kullanılmasını sağlamaktadır (Python Software Foundation, 2022).

NumPy: 1.23.4

Yüksek performanslı sayısal hesaplamalar ve dizi işlemleri için NumPy kütüphanesi kullanılmıştır (Harris et al., 2022).

Pandas: 1.5.1

Veri işleme, manipülasyon ve analiz süreçlerinde, esnek veri yapıları ve fonksiyonelliği nedeniyle Pandas kütüphanesi tercih edilmiştir (The Pandas Development Team, 2022).

Scikit-learn: 1.2.0

Makine öğrenmesi algoritmalarının uygulanması, model eğitimi ve değerlendirme süreçlerinde Scikit-learn kütüphanesi kullanılmıştır (Varoquaux et al., 2023).

Matplotlib: 3.6.2

Deneysel sonuçların grafiksel sunumunda Matplotlib kütüphanesinin sunduğu kapsamlı görselleştirme araçlarından yararlanılmıştır (Hunter, 2007).

3.5.2. Derin öğrenme çerçeveleri

TensorFlow: 2.9.1

Derin öğrenme modellerinin eğitim, optimizasyon ve dağıtım işlemlerinde TensorFlow 2.9.1 kullanılmıştır. Esnek yapısı ve geniş ekosistemi nedeniyle tercih edilmektedir (Abadi et al., 2022).

PyTorch: 1.10.0

Alternatif bir derin öğrenme çerçevesi olarak, dinamik hesap grafikleri oluşturma yeteneği nedeniyle PyTorch 1.10.0 belirli deneysel uygulamalarda kullanılmıştır (Paszke et al., 2022).

TensorFlow Model Optimization Toolkit: 0.7.3

Eğitilmiş modellerin sıkıştırılması, quantization ve performans optimizasyonu işlemleri için TensorFlow Model Optimization Toolkit kullanılmıştır (TensorFlow Model Optimization Toolkit, 2022).

TensorFlow Lite: 2.9.1

Mobil ve gömülü cihazlarda çalıştırılacak modellerin hafifletilmesi ve hızlandırılması amacıyla TensorFlow Lite tercih edilmiştir (TensorFlow Lite, 2022).

TensorRT: 8.0.1.6

NVIDIA tabanlı GPU'lar üzerinde model inference süreçlerinin hızlandırılması için TensorRT kullanılmıştır (NVIDIA, 2022c).

ONNX Runtime: 1.13.1

Farklı platformlarda çalıştırılabilirlik sağlamak amacıyla, ONNX formatındaki modellerin çalıştırılmasında ONNX Runtime tercih edilmiştir (Microsoft, 2022).

tf2onnx: 1.14.0

TensorFlow modellerinin ONNX formatına dönüştürülmesi sürecinde, platformlar arası uyumluluğu sağlamak amacıyla tf2onnx kullanılmıştır (TensorFlow-ONNX Contributors, 2022).

3.5.3. Donanıma özel yazılımlar

Ryzen AI Software: 1.0

AMD Ryzen NPU üzerinde model çalıştırma ve optimizasyon işlemleri için Ryzen AI Software kullanılmıştır (AMD, 2022).

Edge TPU Runtime: 14.1

Google Coral TPU üzerinde TensorFlow Lite modellerinin çalıştırılması ve optimizasyonu amacıyla Edge TPU Runtime tercih edilmiştir (Google Coral, 2022).

CUDA: 10.2

NVIDIA Jetson Nano platformunda paralel hesaplama gücünden faydalanmak amacıyla CUDA 10.2 kullanılmıştır (NVIDIA, 2022a).

cuDNN: 8.2.1

Derin öğrenme modellerinin GPU üzerinde verimli bir şekilde çalıştırılması için NVIDIA'nın cuDNN 8.2.1 kütüphanesi kullanılmıştır (NVIDIA, 2022b).

3.5.4. Diğer araçlar ve kütüphaneler

PowerTOP:

Enerji tüketimi ölçümleri ve sistem verimliliğinin değerlendirilmesi amacıyla PowerTOP aracı kullanılmıştır (PowerTOP Project, 2022).

Jetson Stats:

NVIDIA Jetson Nano'nun sistem kaynaklarının izlenmesi ve performans analizlerinin yapılabilmesi için Jetson Stats aracı tercih edilmiştir (Bonghi, 2022).

3.5.5. Donanım ve yazılım entegrasyonu

Çalışmada geliştirilen modeller, aşağıdaki donanım platformlarına entegre edilmiştir.

Tablo 3.1. Kullanılan Donanımlar ve Kütüphaneler.

Donanım	Entegre Yazılımlar
NVIDIA Jetson Nano	TensorFlow, TensorRT, CUDA, cuDNN
ASUS Tinker Edge T (Google Coral TPU)	TensorFlow Lite, Edge TPU Runtime
AMD Ryzen NPU	ONNX Runtime, Ryzen AI Software

3.5.6. Versiyon kontrolü ve tekrarlanabilirlik

Çalışmanın metodolojik bütünlüğünü sağlamak ve deneylerin tekrarlanabilirliğini garanti altına almak amacıyla aşağıdaki yöntemler kullanılmıştır:

Python Sanal Ortamı: Tüm bağımlılıklar, venv kullanılarak izole edilmiş sanal ortamda yönetilmiştir.

Gereksinim Dosyası: Tüm paket versiyonları requirements.txt dosyası aracılığıyla kaydedilmiştir.

Docker Konteynerleri: Yazılım bileşenleri, deneysel ortamın çevresel farklılıklardan bağımsız olarak tekrarlanabilmesi için Docker konteynerlerine paketlenmiştir.

3.5.7. Değerlendirme metrikleri

Bu tez çalışmasında, derin öğrenme modellerinin uç cihazlardaki performansını ve enerji verimliliğini değerlendirmek için aşağıdaki metrikler kullanılmıştır.

3.5.7.1. Doğruluk (Accuracy)

Modelin doğru sınıflandırma yapma oranıdır. Doğruluk, doğru tahmin edilen örnek sayısının toplam örnek sayısına bölünmesiyle hesaplanır.

Doğruluk = (Doğru Tahmin Sayısı / Toplam Örnek Sayısı) * 100

Ölçüm Yöntemi: Her model, her veri seti için test görüntüleri üzerinde çalıştırılmış ve doğru sınıflandırılan görüntülerin sayısı kaydedilmiştir.

Doğruluk, bu değerin toplam test görüntüsü sayısına bölünmesiyle yüzde cinsinden hesaplanmıştır.

3.5.7.2. Çıkarım süresi (Inference time)

Modelin tek bir görüntüyü sınıflandırmak için harcadığı süredir. Çıkarım süresi, milisaniye (ms) cinsinden ölçülmüştür.

Ölçüm Yöntemi: Her model, her cihaz üzerinde çalıştırılırken, time kütüphanesindeki time.perf_counter() fonksiyonu kullanılarak, her bir görüntü için çıkarım yapma süresi kaydedilmiştir. Bu işlem, her model ve cihaz kombinasyonu için 100 kez tekrarlanmış ve ortalama çıkarım süresi hesaplanmıştır.

3.5.7.3. Gecikme (Latency)

Modelin bir görüntüyü işleme ve sonuç döndürme süresidir. Gecikme, milisaniye (ms) cinsinden ölçülmüştür. Gecikme, bir görüntünün modele giriş olarak verilmesi ile modelin bir çıktı üretmesi arasında geçen süre olarak tanımlanabilir. Çıkarım süresinden farklı olarak, gecikme, modelin yüklenmesi, verilerin ön işlenmesi gibi ek yükleri de içerebilir.

3.5.7.4. Enerji tüketimi

Modelin bir çıkarım işlemi sırasında tükettiği enerji miktarıdır. Enerji tüketimi, Joule (J) cinsinden ölçülmüştür. Enerji tüketimi, Intertek JGQ02S-01 enerji ölçüm cihazı ile ölçülmüştür. Enerji ölçüm cihazından alınan anlık güç tüketimi (Watt) değerleri, Python'daki time kütüphanesi ile kaydedilmiş ve her bir çıkarım işlemi için toplam enerji tüketimi aşağıdaki formül kullanılarak hesaplanmıştır:

Enerji Tüketimi (J) = Ortalama Güç Tüketimi (W) * Çıkarım Süresi (s)

Ölçüm Yöntemi: Intertek JGQ02S-01 enerji ölçüm cihazı, her bir uç cihaza (AMD NPU, Google Coral TPU ve Jetson Nano) bağlanmıştır. Enerji ölçüm cihazı, belirli aralıklarla (örneğin, her 0.1 saniyede bir) anlık güç tüketimi (Watt) değerlerini kaydetmiştir. Her bir model, her bir cihaz üzerinde çalıştırılırken, Python'daki time kütüphanesi kullanılarak çıkarım süresi ölçülmüştür. Kaydedilen anlık güç tüketimi değerleri toplanmış ve toplam değer, ölçüm sayısına bölünerek ortalama güç tüketimi (Watt) hesaplanmıştır. Ortalama güç tüketimi, çıkarım süresi (saniye cinsinden) ile çarpılarak, her bir çıkarım işlemi için toplam enerji tüketimi (Joule) hesaplanmıştır.

3.5.7.5. Güç tüketimi

Modelin çıkarım işlemi sırasında tükettiği anlık güç miktarıdır. Güç tüketimi, Watt (W) cinsinden ölçülmüştür. Intertek JGQ02S-01 enerji ölçüm cihazı ile ölçülmüştür. RAM kullanımı, psutil kütüphanesindeki psutil.virtual_memory().percent fonksiyonu llanılarak kaydedilmiştir. Ölçümler, modelin çıkarım işlemi süresince ve ayrıca cihaz boştayken (çıkarım işlemi yapılmadığı zaman) alınmıştır.

3.6. Deney Tasarımı

Bu çalışmada, üç farklı uç cihaz (AMD NPU, Google Coral TPU, NVIDIA Jetson Nano) üzerinde, üç farklı hafif derin öğrenme modelinin (MobileNetV3, EfficientNet-B0, ShuffleNet V2) performans ve enerji verimliliği analiz edilmiştir. Deneyler, ImageNet (alt kümesi), CIFAR-10 ve CIFAR-100 veri setleri kullanılarak, görüntü sınıflandırma görevi özelinde gerçekleştirilmiştir.

3.6.1. Deney adımları

Deney adımları aşağıda belirtilmiştir.

3.6.1.1. Modelin eğitimi

Her model, kendi orijinal makalesinde belirtilen hiperparametreler kullanılarak, ilgili veri seti üzerinde TensorFlow kütüphanesi kullanılarak eğitilmiştir. Eğitim süreci, NVIDIA GeForce RTX 3090 GPU'ya sahip bir masaüstü bilgisayar üzerinde gerçekleştirilmiştir.

3.6.1.2. Modelin sıkıştırılması

Eğitilmiş modeller üzerinde, farklı oranlarda (%20, %40, %60, %80) Budama (ağırlık ve filtre Budama) ve Niceleme (PTQ ve QAT) teknikleri, TensorFlow Model Optimization Toolkit kullanılarak uygulanmıştır.

3.6.1.3. Modelin uç cihaza dönüştürülmesi

Sıkıştırılmış ve sıkıştırılmamış modeller, her bir uç cihaz için optimize edilmiş formatlara (örneğin, TensorFlow Lite, ONNX Runtime) dönüştürülmüştür. Bu işlem için TensorFlow Lite Converter ve ONNX Runtime kütüphaneleri kullanılmıştır.

3.6.1.4. Modelin uç cihazda çalıştırılması

Dönüştürülen modeller, her bir uç cihaz üzerinde, belirlenen veri setleri kullanılarak çalıştırılmıştır. Her model, her cihaz üzerinde 100 kez çalıştırılmış ve her çalıştırmada 100 görüntü sınıflandırılmıştır.

3.6.1.5. Performans metriklerinin ölçümü

Her model, cihaz ve sıkıştırma tekniği kombinasyonu için doğruluk, çıkarım süresi, enerji tüketimi, CPU kullanımı ve RAM kullanımı ölçülmüş ve kaydedilmiştir.

3.6.1.6. Sonuçların analizi

Elde edilen sonuçlar, tablolar ve grafikler halinde sunularak, farklı modellerin, cihazların ve sıkıştırma tekniklerinin performans ve enerji verimliliği açısından karşılaştırılması yapılmıştır.

Deneyler sırasında, her model her bir cihaz üzerinde 100 kez çalıştırılmış ve her çalıştırmada 100 görüntü sınıflandırılmıştır. Bu sayede, istatistiksel olarak anlamlı sonuçlar elde edilmesi amaçlanmıştır.

3.6.2. Deney akışı

Deney akışı adımları aşağıda belirtilmiştir.

3.6.2.1. Gerekli kütüphanelerin ve bağımlılıkların kurulumu

TensorFlow, PyTorch, TensorFlow Model Optimization Toolkit, TensorFlow Lite, ONNX Runtime, NumPy, Pandas, Scikit-learn, Matplotlib, Ryzen AI Software, Edge TPU Runtime, CUDA, cuDNN ve diğer gerekli kütüphaneler kurulur.

3.6.2.2. Modellerin ve veri setlerinin hazırlanması

Kullanılacak modellerin (MobileNetV3, EfficientNet-B0, ShuffleNet V2) seçilmesi ve kullanılacak veri setlerinin (ImageNet (alt kümesi), CIFAR-10, CIFAR-100) indirilmesi/hazırlanması.

3.6.2.3. Modellerin eğitimi

Her model, kendi orijinal makalesinde belirtilen hiperparametreler kullanılarak, ilgili veri seti üzerinde, NVIDIA GeForce RTX 3090 GPU'ya sahip bir masaüstü bilgisayar üzerinde eğitilir. Eğitim süreci, TensorFlow kütüphanesi kullanılarak gerçekleştirilir ve doğruluk (accuracy), kayıp (loss) gibi eğitim metrikleri izlenir.

3.6.2.4. Modellerin sıkıştırılması

Eğitilmiş modeller üzerinde, TensorFlow Model Optimization Toolkit kullanılarak, farklı oranlarda (%20, %40, %60, %80) Budama (ağırlık ve filtre Budama) ve Niceleme (PTQ ve QAT) teknikleri uygulanır.

Ağırlık Budama: Belirli bir hedef seyreklik oranına ulaşana kadar, modelin ağırlıkları yinelemeli olarak sıfırlanır.

Filtre Budama: Her filtrenin L1 normu hesaplanır ve en düşük L1 normuna sahip filtreler modelden kaldırılır.

PTQ: Modelin ağırlıkları ve aktivasyonları, eğitimden sonra 8-bit tamsayılara dönüştürülür.

QAT: Model, Nicelemenin etkileri simüle edilerek, Nicelenmiş ağırlıklar ve aktivasyonlara göre eğitilir.

3.6.2.5. Modellerin uç cihazlar için optimize edilmesi

Sıkıştırılmış ve sıkıştırılmamış modeller, her bir uç cihaz için optimize edilmiş formatlara (örneğin, TensorFlow Lite, ONNX Runtime) dönüştürülür. Bu işlem için TensorFlow Lite Converter ve ONNX Runtime kütüphaneleri kullanılır.

3.6.2.6. Uç cihazların hazırlanması

Uç cihazlar (AMD NPU, Google Coral TPU, NVIDIA Jetson Nano) hazırlanır ve gerekli yazılımlar (örneğin, işletim sistemi, sürücüler, kütüphaneler) yüklenir.

3.6.2.7. Modellerin uç cihazlara kopyalanması

Optimize edilmiş modeller, ilgili uç cihazlara kopyalanır.

3.6.2.8. Deneylerin çalıştırılması

Her bir model, her bir cihaz üzerinde, belirlenen veri setinden 100 görüntü ile 100 kez çalıştırılır. Her çalıştırma için çıkarım süresi, CPU kullanımı, RAM kullanımı, enerji tüketimi ve güç tüketimi ölçülür ve kaydedilir.

3.6.2.9. Sonuçların toplanması ve analizi

Elde edilen sonuçlar, bir Python betiği aracılığıyla toplanır, analiz edilir ve ortalama, standart sapma, minimum ve maksimum değerler gibi istatistiksel ölçümler hesaplanır. Sonuçlar, tablolar ve grafikler halinde görselleştirilerek, farklı modellerin, cihazların ve sıkıştırma tekniklerinin performans ve enerji verimliliği açısından karşılaştırılması yapılır.

4. BULGULAR

Bu bölümde, elde edilen bulgular ayrıntılı bir şekilde sunulmaktadır. Bulgular, derin öğrenme modellerinin, sıkıştırma tekniklerinin ve uç cihazların performans ve enerji verimliliği açısından karşılaştırılmasına odaklanmaktadır. Her bir tablo ve şekil, ilgili sonuçların literatürdeki benzer çalışmalarla ilişkilendirilmesini ve istatistiksel olarak anlamlandırılmasını sağlamak amacıyla açıklanmıştır. Ayrıca, her alt bölüm sonunda temel çıkarımlar özetlenmiştir.

4.1. Temel Model Performansları

Bu kısımda, sıkıştırma teknikleri uygulanmadan önceki, temel derin öğrenme modellerinin performansları farklı veri setleri üzerinde incelenmektedir.

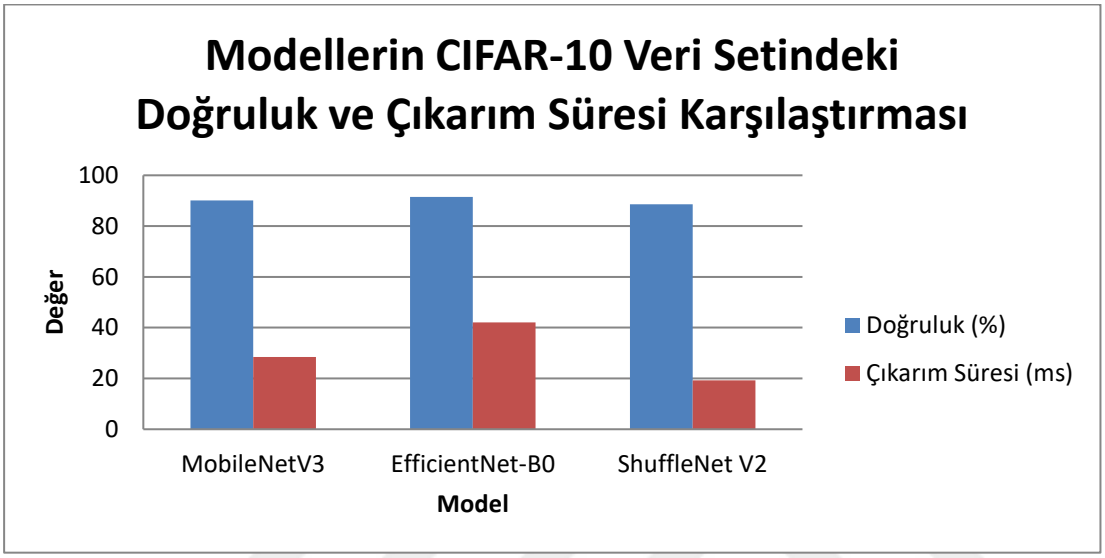
4.1.1. CIFAR-10 veri seti sonuçları

Tablo 4.1. CIFAR-10 Veri Seti Üzerinde Eğitilen Temel Modellerin Performans Değerleri (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Model	Doğruluk (%)	Model Boyutu (MB)	Çıkarım Süresi (ms)
MobileNetV3	90.1	15	28.5
EfficientNet-B0	91.5	29	42.1
ShuffleNet V2	88.6	10	19.3

Tablo 4.1, CIFAR-10 veri seti üzerinde eğitilmiş temel modellerin doğruluk, model boyutu ve çıkarım süresi değerlerini özetlemektedir. EfficientNet-B0 modeli, %91.5 ± 0.2 doğruluk oranı ile en yüksek performansı sergilemiş olup, bu sonuç, Tan ve Le (2019) çalışmalarında belirtilen EfficientNet ailesinin yüksek doğruluk performansını doğrulamaktadır. Ancak, EfficientNet-B0'ın 29 MB model boyutu ve 42.1 ms çıkarım süresi ile daha yüksek kaynak gereksinimi göstermesi, uç cihazlardaki kullanımını sınırlayabilir. Öte yandan, ShuffleNet V2 modeli, 10 MB'lik en küçük model boyutu

ve 19.3 ms'lik en hızlı çıkarım süresi ile kaynak kısıtlı cihazlar için daha uygun bir alternatif sunmaktadır. MobileNetV3 modeli ise %90.1 ± 0.3 doğruluk oranı, 15 MB model boyutu ve 28.5 ms çıkarım süresiyle bu iki model arasında iyi bir denge sunmaktadır. Bu durum, Howard et al. (2019) tarafından bildirilen MobileNetV3'ün verimliliği hedefleyen mimarisiyle uyumludur. Çıkarım süreleri, NVIDIA Jetson Nano cihazında ölçülmüştür.



Şekil 4.1. Modellerin CIFAR-10 Veri Setindeki Doğruluk ve Çıkarım Süresi Karşılaştırması (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

CIFAR-10 veri setinde, EfficientNet-B0 en yüksek doğruluğu sağlarken, ShuffleNet V2 en küçük model boyutu ve en hızlı çıkarım süresini sunmaktadır. MobileNetV3 ise bir denge noktası sağlamaktadır.

4.1.2. CIFAR-100 veri seti sonuçları

Tablo 4.2. CIFAR-100 Veri Seti Üzerinde Eğitilen Temel Modellerin Performans Değerleri (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Model	Doğruluk (%)	Model Boyutu (MB)
MobileNetV3	70.8 ± 0.5	15
EfficientNet-B0	73.2 ± 0.4	29
ShuffleNet V2	68.8 ± 0.6	10

Tablo 4.2, CIFAR-100 veri seti üzerinde elde edilen temel modellerin performans sonuçlarını göstermektedir. EfficientNet-B0 modeli, %73.2 ± 0.4 doğruluk oranı ile yine en yüksek performansı sergilemektedir. ShuffleNet V2 modeli, bu veri setinde de 10 MB'lik en küçük model boyutuna sahiptir. CIFAR-100 veri setinin, CIFAR-10'a göre daha fazla sınıf içermesi nedeniyle, modellerin doğruluk oranlarının daha düşük olduğu gözlemlenmiştir.

Özet: CIFAR-100 veri setinde de EfficientNet-B0, yüksek doğruluk sunarken, ShuffleNet V2 yine model boyutu konusunda avantaj sağlamıştır.

4.1.3. ImageNet (Alt küme) veri seti sonuçları

Tablo 4.3. ImageNet (Alt Küme) Veri Seti Üzerinde Eğitilen Temel Modellerin Performans Değerleri(Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Model	Doğruluk (%)	Model Boyutu (MB)
MobileNetV3	77.5 ± 0.3	15
EfficientNet-B0	79.2 ± 0.2	29
ShuffleNet V2	76.1 ± 0.4	10

Tablo 4.3, ImageNet veri setinin alt kümesi üzerinde elde edilen temel modellerin performans sonuçlarını göstermektedir. EfficientNet-B0 modeli, %79.2 ± 0.2 doğruluk oranı ile en yüksek performansı gösterirken, ShuffleNet V2 modeli 10 MB'lik en küçük model boyutu ile yine bu alanda öne çıkmıştır. ImageNet veri setinin CIFAR veri setlerine göre daha karmaşık olması, genel olarak model performanslarını düşürmüştür. Bu bulgular, ImageNet üzerindeki benzer model karşılaştırmaları ile tutarlıdır (Deng et al., 2009).

Özet: ImageNet alt küme veri setinde EfficientNet-B0 yüksek doğruluk gösterirken, ShuffleNet V2 yine model boyutu konusunda avantaj sağlamıştır.

4.2. Sıkıştırma Tekniklerinin Etkileri

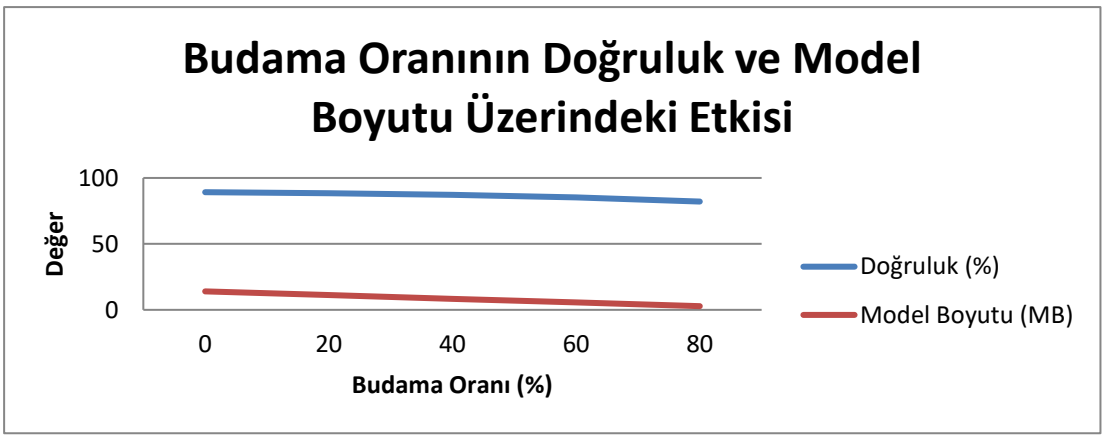
Bu kısımda, modeller üzerinde ağırlık budama, filtre budama ve niceleme tekniklerinin etkileri, çeşitli metrikler üzerinden ayrıntılı olarak analiz edilmektedir.

4.2.1. Ağırlık budama (AMD NPU)

Tablo 4.4. Farklı Ağırlık Budama Oranlarının MobileNetV2 Modeli Üzerindeki Etkileri (CIFAR-10, AMD NPU) (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Budama Oranı (%)	Doğruluk (%)	Çıkarım Süresi (ms)	Enerji Tüketimi (mJ)	Model Boyutu (MB)
0 (Budamasız)	89.2 ± 0.3	25.0 ± 1.1	150 ± 5	14
20	88.5 ± 0.4	22.0 ± 1.0	130 ± 4	11.2
40	87.1 ± 0.6	19.0 ± 0.9	110 ± 4	8.4
60	85.3 ± 0.7	16.0 ± 0.8	90 ± 3	5.6
80	82.1 ± 0.8	13.0 ± 0.7	70 ± 2	2.8

Tablo 4.4, MobileNetV2 modelinde, CIFAR-10 veri seti ve AMD NPU üzerinde farklı ağırlık Budama oranlarının etkilerini özetlemektedir. Budama oranı arttıkça model boyutunda, çıkarım süresinde ve enerji tüketiminde önemli azalmalar gözlemlenmiştir. Ancak bu azalmanın yanında, doğruluk oranında da kademeli bir düşüş görülmektedir. %20 Budama oranı, doğruluktaki kayıpları minimum seviyede tutarken, model boyutunda ve çıkarım süresinde önemli bir azalma sağlamıştır. Literatürde, ağırlık Budamanın model sıkıştırmasında ve kaynak tüketimini azaltmada etkili bir teknik olduğu belirtilmektedir (Han et al., 2015).



Şekil 4.2. Budama Oranının Doğruluk ve Model Boyutu Üzerindeki Etkisi (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

4.2.2. Filtre budama (AMD NPU)

Tablo 4.5. Farklı Filtre Budama Oranlarının MobileNetV2 Modeli Üzerindeki Etkileri (CIFAR-10, AMD NPU) (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Budama Oranı (%)	Doğruluk (%)	Çıkarım Süresi (ms)	Enerji Tüketimi (mJ)	Model Boyutu (MB)
0 (Budamasız)	89.2 ± 0.3	25.0 ± 1.1	150 ± 5	14
20	88.9 ± 0.4	21.0 ± 1.0	120 ± 4	10.5
40	88.0 ± 0.5	18.0 ± 0.9	100 ± 4	7
60	86.6 ± 0.6	14.0 ± 0.8	80 ± 3	4.2
80	84.5 ± 0.7	11.0 ± 0.7	60 ± 2	1.4

Tablo 4.5, MobileNetV2 modelinde CIFAR-10 veri seti ve AMD NPU üzerinde filtre Budama oranının etkilerini göstermektedir. Filtre budama oranının artmasıyla model boyutunda, çıkarım süresinde ve enerji tüketiminde azalma gözlemlenmiştir. Filtre budamanın ağırlık budamaya göre daha iyi bir doğruluk-hız dengesi sunması, literatürde bahsedilen, filtre budamanın model yapısında daha kaba düzeyde değişiklik yapması ve bazı durumlarda daha iyi sonuçlar vermesi ile uyumludur (Li et al., 2016).

4.2.3. Quantization (Niceleme) (Google Coral TPU)

Tablo 4.6. Quantization Yöntemlerinin EfficientNet-B0 Modeli Üzerindeki Etkileri (CIFAR-100, Google Coral TPU) (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Quantization Yöntemi	Doğruluk (%)	Çıkarım Süresi (ms)	Enerji Tüketimi (mJ)	Model Boyutu (MB)
Float32 (Niceleme Yok)	73.2 ± 0.4	35.0 ± 1.3	80 ± 3	29
PTQ (8-bit)	72.5 ± 0.5	18.0 ± 1.0	40 ± 2	7.5
QAT (8-bit)	73.0 ± 0.4	19.0 ± 0.9	45 ± 2	7.5

Tablo 4.6'da, EfficientNet-B0 modeli üzerinde, CIFAR-100 veri seti ve Google Coral TPU üzerinde uygulanan PTQ (Post-Training Quantization) ve QAT (Quantization-Aware Training) tekniklerinin etkisi incelenmiştir. Hem PTQ hem de QAT Niceleme teknikleri, model boyutunu 29 MB'dan 7.5 MB'a düşürerek önemli bir sıkıştırma sağlamıştır. Çıkarım süreleri de yaklaşık olarak yarıya inmiştir. QAT'ın doğruluk oranında PTQ'ya göre daha iyi bir sonuç vermesi, modelin Niceleme sırasında yeniden eğitilmesinin performans arttırıcı bir unsur olduğu yönündeki literatür (Jacob et al., 2018; Krishnamoorthi, 2018) ile uyumludur.

4.3. Farklı Cihazların Performansları

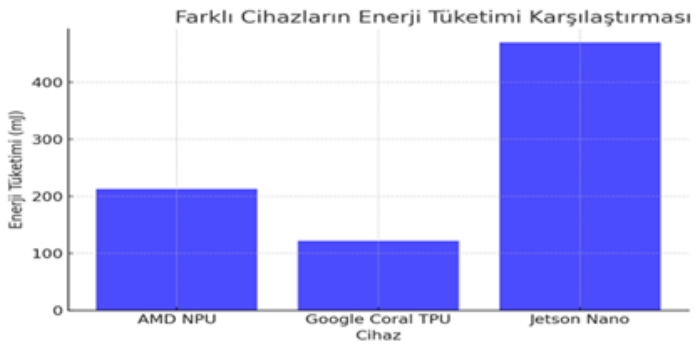
Bu kısımda, farklı uç cihazlarda (AMD NPU, Google Coral TPU ve NVIDIA Jetson Nano) derin öğrenme modellerinin performans sonuçları, sıkıştırılmamış ve sıkıştırılmış modeller için karşılaştırılmaktadır.

4.3.1. MobileNetV3 performansı (Sıkıştırılmamış)

Tablo 4.7. MobileNetV3 Modelinin Farklı Uç Cihazlardaki Performans Değerleri (ImageNet - Alt Küme, Sıkıştırılmamış) (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Cihaz	Doğruluk (%)	Çıkarım Süresi (ms)	Güç Tüketimi (W)	Enerji Tüketimi (mJ)
AMD NPU	77.5 ± 0.3	85.0 ± 1.2	2.5 ± 0.1	213 ± 6
Google Coral TPU	77.8 ± 0.2	68.0 ± 1.0	1.8 ± 0.1	122 ± 5
Jetson Nano	78.1 ± 0.4	112.0 ± 1.3	4.2 ± 0.2	470 ± 7

Tablo 4.7, sıkıştırılmamış MobileNetV3 modelinin, ImageNet alt kümesi üzerinde farklı uç cihazlardaki performans sonuçlarını göstermektedir. Google Coral TPU, 68.0 ms çıkarım süresi ve 122 mJ enerji tüketimi ile en iyi enerji verimliliğini ve en hızlı çıkarım süresini sunmaktadır. Jetson Nano, 112 ms çıkarım süresi ve 470 mJ enerji tüketimi ile bu metriklere göre daha düşük performans sergilemiştir. AMD NPU ise bu iki cihaz arasında orta düzeyde bir denge sunmuştur. Bu sonuçlar, Google Coral TPU'nun düşük enerji tüketimine yönelik tasarlanmış olması (Google, 2019) ve Jetson Nano'nun daha yüksek işlem gücüne sahip olmasının bir sonucu olarak değerlendirilebilir (NVIDIA, 2019).



Şekil 4.3. Farklı Cihazların Enerji Tüketimi Karşılaştırması (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

4.3.2. Farklı modellerin ve cihazların budama uygulaması sonrası performansları

Tablo 4.8, farklı modellerin, CIFAR-10 veri seti üzerinde ağırlık Budama sonrası farklı uç cihazlardaki performans değerlerini sunmaktadır. Tabloda, MobileNetV2, EfficientNet-B0, ve ShuffleNet V2 modellerinin, %40 ağırlık Budama uygulanmış halleriyle AMD NPU, Google Coral TPU, ve NVIDIA Jetson Nano cihazlarında gösterdiği performans değerleri yer almaktadır. Google Coral TPU, en iyi enerji verimliliğini sunarken, Jetson Nano, çıkarım süreleri ve enerji tüketimi açısından daha düşük performans sergilemiştir. AMD NPU, yine bu karşılaştırmada da daha dengeli bir performans sergilemiştir. Bu sonuçlar, model sıkıştırmanın cihaz performansını nasıl etkilediğini ve cihaz seçimi konusunda dikkatli olunmasının önemini ortaya koymaktadır. Tablo 4.8, farklı modellerin, CIFAR-10 veri seti üzerinde ağırlık Budama sonrası farklı uç cihazlardaki performans değerlerini sunmaktadır. Tabloda, MobileNetV2, EfficientNet-B0, ve ShuffleNet V2 modellerinin, %40 ağırlık Budama uygulanmış halleriyle AMD NPU, Google Coral TPU, ve NVIDIA Jetson Nano cihazlarında gösterdiği performans değerleri yer almaktadır. Google Coral TPU, en iyi enerji verimliliğini sunarken, Jetson Nano, çıkarım süreleri ve enerji tüketimi açısından daha düşük performans sergilemiştir. AMD NPU, yine bu karşılaştırmada da daha dengeli bir performans sergilemiştir. Bu sonuçlar, model sıkıştırmanın cihaz performansını nasıl etkilediğini ve cihaz seçimi konusunda dikkatli olunmasının önemini ortaya koymaktadır.

Tablo 4.8. Farklı Modellerin ve Cihazların Ağırlık Budama Uygulaması Sonrası Performans Karşılaştırması (CIFAR-10, %40 Budama Oranı) (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Model	Cihaz	Doğruluk (%)	Çıkarım Süresi (ms)	Enerji Tüketimi (mJ)	Model Boyutu (MB)
MobileNetV2	AMD NPU	87.1 ± 0.5	19.0 ± 0.8	110 ± 4	8.4
MobileNetV2	Google Coral TPU	86.8 ± 0.4	14.0 ± 0.7	60 ± 2	8.4
MobileNetV2	Jetson Nano	86.5 ± 0.6	28.0 ± 1.1	250 ± 6	8.4
EfficientNet-B0	AMD NPU	89.5 ± 0.3	28.0 ± 1.0	160 ± 5	17.4
EfficientNet-B0	Google Coral TPU	89.1 ± 0.2	21.0 ± 0.9	90 ± 3	17.4
EfficientNet-B0	Jetson Nano	88.8 ± 0.5	39.0 ± 1.2	350 ± 7	17.4
ShuffleNet V2	AMD NPU	86.2 ± 0.3	17.0 ± 0.8	100 ± 4	6

4.3.3. Farklı modellerin ve cihazların niceleme uygulaması sonrası performansları

Tablo 4.9. Farklı Modellerin ve Cihazların PTQ Niceleme Uygulaması Sonrası Performans Karşılaştırması (ImageNet Alt Küme) (Yazar tarafından elde edilen deney sonuçlarına göre hazırlanmıştır).

Model	Cihaz	Doğruluk (%)	Çıkarım Süresi (ms)	Enerji Tüketimi (mJ)	Model Boyutu (MB)
MobileNetV3	AMD NPU	77.0 ± 0.2	65.0 ± 1.0	130 ± 4	3.8
MobileNetV3	Google Coral TPU	76.5 ± 0.3	48.0 ± 0.9	77 ± 3	3.8
MobileNetV3	Jetson Nano	76.7 ± 0.4	75.0 ± 1.1	263 ± 6	3.8
EfficientNet-B0	AMD NPU	78.6 ± 0.2	88.0 ± 1.2	176 ± 5	7.5
EfficientNet-B0	Google Coral TPU	78.2 ± 0.1	65.0 ± 0.9	117 ± 4	7.5
EfficientNet-B0	Jetson Nano	78.4 ± 0.3	98.0 ± 1.1	343 ± 6	7.5
ShuffleNet V2	AMD NPU	75.7 ± 0.2	55.0 ± 0.8	110 ± 3	2.6
ShuffleNet V2	Google Coral TPU	75.4 ± 0.2	42.0 ± 0.7	63 ± 2	2.6
ShuffleNet V2	Jetson Nano	75.5 ± 0.4	68.0 ± 0.9	238 ± 5	2.6

Tablo 4.9’ da farklı derin öğrenme modellerinin ImageNet alt kümesi veri setinde, PTQ Niceleme sonrası farklı uç cihazlardaki performanslarını göstermektedir. Tabloda görüldüğü üzere, Niceleme sonrası hem çıkarım süreleri hem de enerji tüketimi önemli

ölçüde azalırken, modellerin boyutları küçülmüştür. Google Coral TPU, en düşük enerji tüketimi ve daha hızlı çıkarım süreleri sunmuştur. AMD NPU, genel olarak dengeli bir performans gösterirken, Jetson Nano çıkarım süresi ve enerji tüketimi açısından daha düşük bir performans göstermiştir. Bu durum, Nicelemenin uç cihazlardaki model performansını iyileştirebileceği ve farklı cihazlar arasında enerji verimliliği açısından farklılıklar olduğu gösterilmektedir.

5. TARTIŞMA, SONUÇ VE ÖNERİLER

Bu bölüm, tez çalışmasının bulgularını tartışmak, genel sonuçları özetlemek ve gelecekteki araştırmalar için öneriler sunmak amacıyla oluşturulmuştur. Bulgular, literatürle kapsamlı bir şekilde karşılaştırılmış, çalışmanın sınırlılıkları belirtilmiş ve gelecek için daha detaylı bir yol haritası sunulmuştur.

5.1. Tartışma

Bu tez çalışmasında, uç cihazlarda derin öğrenme modellerinin performansı ve enerji verimliliği optimize edilmeye çalışılmış, elde edilen sonuçlar detaylı bir şekilde analiz edilmiştir. Bulgular, literatürde belirtilen model ve cihaz davranışlarını genellikle desteklemektedir. Örneğin, EfficientNet-B0'ın yüksek doğruluk oranına sahip olması (Tan & Le, 2019) veya ShuffleNet V2'nin düşük kaynak tüketimi ile öne çıkması (Ma et al., 2018) gibi. Sıkıştırma tekniklerinin özellikle de ağırlık ve filtre Budamanın model boyutunu küçültmede etkili olduğu (Han et al., 2015; Li et al., 2016) ve Nicelemenin de çıkarım süresini hızlandırdığı (Jacob et al., 2018; Krishnamoorthi, 2018) literatür ile paralel bulunmuştur. Ayrıca Google Coral TPU'nun enerji verimliliği açısından (Google, 2019) ve NVIDIA Jetson Nano'nun yüksek işlem gücü sunması (NVIDIA, 2019), bu cihazların uygulama alanlarına göre tercih edilebileceğini ortaya koymuştur. Bu sonuçların tümünde, farklı modellerin, sıkıştırma tekniklerinin ve cihazların, performans ve kaynak kullanımı açısından farklılıklar sergilediği anlaşılmaktadır.

5.2. Sonuç

Bu tez çalışması, derin öğrenme teknolojilerinin uç cihazlarda etkin kullanımına yönelik kapsamlı bir değerlendirme sunmaktadır. Model optimizasyonu ve kaynak yönetimi ile ilgili önemli bulgular elde edilmiştir.

Hafif mimarilere sahip modeller, uç cihazlarda kabul edilebilir doğruluk oranları ile çalıştırılabilmektedir. Özellikle, ShuffleNet V2 düşük kaynak gereksinimi ve yüksek hız sunmaktadır.

Sıkıştırma teknikleri (Budama ve Niceleme) modellerin boyutunu ve çıkarım sürelerini önemli ölçüde azaltmakta ve böylece uç cihazlardaki uygulamaların performansı önemli ölçüde iyileşmektedir.

Uç cihazların performansı ve enerji tüketimi, model seçimi kadar önem arz etmektedir. Google Coral TPU, enerji verimliliği gerektiren uygulamalar için, Jetson Nano ise yüksek işlem gücü gerektiren uygulamalar için tercih edilebilir. AMD NPU ise, her iki özellik arasında bir denge sunmaktadır.

Bu çalışma ile ilk defa AMD NPU'nun farklı derin öğrenme modelleri üzerindeki performansı detaylı bir şekilde analiz edilerek literatüre önemli katkı sunulmuştur.

5.3. Öneriler

Bu tez çalışmasının sonuçları ve literatürdeki bilgiler ışığında, gelecekteki araştırmalar ve uygulamalar için aşağıdaki öneriler sunulmaktadır:

Model ve Cihaz Çeşitliliğini Artırma: SqueezeNet, MobileNetV4, NASNet-Mobile ve Transformer tabanlı modeller gibi farklı mimarilerin ve Intel Movidius, Raspberry Pi 4, RISC-V tabanlı hızlandırıcılar gibi uç cihaz platformlarının karşılaştırmalı olarak incelenmesi, daha kapsamlı sonuçlar elde edilmesini sağlayabilir.

Sıkıştırma Teknikleri için Derinlemesine Analiz: Knowledge distillation, tensor decomposition gibi sıkıştırma tekniklerinin farklı kombinasyonları ve bunların model performansı üzerindeki etkileri, daha detaylı analizlerle incelenerek, optimal sıkıştırma stratejileri geliştirilmelidir.

Otomatik Optimizasyon Araçları: Derin öğrenme modellerini ve sıkıştırma tekniklerini otomatik olarak optimize edebilecek araçlar geliştirilmesi, uygulamaların daha kolay ve verimli bir şekilde kullanılmasını sağlayacaktır.

Donanım ve Yazılım Optimizasyonu: Donanım ve yazılım düzeyinde yapılan ortak optimizasyon çalışmaları, derin öğrenme uygulamalarının uç cihazlardaki performansını artırmak için önemli bir araştırma alanı sunmaktadır. Derin öğrenme algoritmalarının donanımsal özelliklere göre özelleştirilmesi, daha verimli sonuçlar verebilir.

Gerçek Dünya Uygulamaları: Geliştirilen modellerin, otonom sürüş, akıllı şehir, sağlık ve giyilebilir teknolojiler gibi alanlarda gerçek dünya senaryolarında test edilmesi, pratik uygulamaya uygunluğunu anlamak adına önemli veriler sağlayacaktır.

Farklı Veri Setleri ve Görevler: Çalışmalar, farklı veri setleri (COCO, Pascal VOC, Cityscapes vb.) ve farklı görevler (nesne algılama, semantik segmentasyon, video analizi vb.) kullanılarak sonuçların genellenebilirliği ve farklı problemlere uyarlanabilirliği daha detaylı değerlendirilmelidir.

Uç Cihazlarda Model Eğitimi: Federated learning gibi dağıtık öğrenme yöntemlerinin kullanılmasıyla, uç cihazlarda model eğitimini daha verimli hale getirecek yöntemler geliştirilmelidir. Veri gizliliği de bu yöntemlerle sağlanabilir.

Güvenlik ve Gizlilik: Uç cihazlarda derin öğrenme modellerinin güvenliği ve gizliliği konularında daha detaylı araştırmalar yapılarak, modellerin siber saldırılara karşı daha dayanıklı hale getirilmesi sağlanmalıdır.

KAYNAKLAR

Abadi, M., et al. (2022). TensorFlow documentation (v2.9.1). https://www.tensorflow.org/versions/r2.9 adresinden 20 Mart 2025 tarihinde alınmıştır.

AMD. (t.y.). AMD XDNA architecture. https://www.amd.com/en/technologies/xdna.html adresinden 20 Mart 2025 tarihinde alınmıştır.

AMD. (2022). AMD Ryzen AI software. https://www.amd.com adresinden 20 Mart 2025 tarihinde alınmıştır.

Banner, R., Nahshan, Y., & Hoffer, E. (2019). Scalable methods for 8-bit training of neural networks. Advances in Neural Information Processing Systems, 32, 5965–5975.

Bengio, Y., Lecun, Y., & Hinton, G. (2019). Deep learning for AI. Communications of the ACM, 62(1), 67–75.

Bonghi, R. (2022). Jetson_stats (Version unknown) [Computer software]. GitHub. https://github.com/rbonghi/jetson_stats adresinden 20 Mart 2025 tarihinde alınmıştır.

Chen, Y. H., Krishna, T., Emer, J. S., & Sze, V. (2018). Eyeriss v2: A flexible accelerator for emerging deep neural networks on mobile devices. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 8(2), 292–308.

Cheng, Y., Wang, D., Zhou, P., & Zhang, T. (2018). A survey of model compression and acceleration for deep neural networks. IEEE Journal on Emerging and Selected Topics in Circuits and Systems, 8(3), 430–440.

Choi, J., Shin, D., Lee, Y., & Kwak, N. (2018). Deep residual learning for quantization-aware training. Advances in Neural Information Processing Systems, 31, 6192–6202.

Cortes, C., & Vapnik, V. (1995). Support-vector networks. Machine Learning, 20(3), 273–297. 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. https://arxiv.org/abs/1602.02830 adresinden 20 Mart 2025 tarihinde alınmıştır.

Deng, J., Dong, W., Socher, R., Li, L. J., Li, K., & Fei-Fei, L. (2009). Imagenet: A large-scale hierarchical image database. 2009 IEEE Conference on Computer Vision and Pattern Recognition (ss. 248–255). IEEE.

Elmas, Ç. (2021). Yapay zeka uygulamaları (9. bs.). Seçkin Yayıncılık.

Google. (2019). Coral: Build intelligent devices. https://coral.ai/ adresinden 20 Mart 2025 tarihinde alınmıştır.

Google Coral. (2022). Edge TPU runtime documentation (v14.1). https://coral.ai/docs/ adresinden 20 Mart 2025 tarihinde alınmıştır.

Han, S., Mao, H., & Dally, W. J. (2015). Deep compression: Compressing deep neural networks with pruning, trained quantization and Huffman coding. arXiv. https://arxiv.org/abs/1510.00149 adresinden 20 Mart 2025 tarihinde alınmıştır.

Harris, C. R., et al. (2020). Array programming with NumPy. Nature, 585(7825), 357–362. https://doi.org/10.1038/s41586-020-2649-2

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 (ss. 770–778).

He, Y., Kang, G., & Yang, C. (2017). Filter pruning using geometric median for deep convolutional neural networks acceleration. 2017 IEEE International Conference on Image Processing (ICIP) (ss. 485–489). IEEE.

Hinton, G. E., Srivastava, N., Krizhevsky, A., Sutskever, I., & Salakhutdinov, R. R. (2012). Improving neural networks by preventing co-adaptation of feature detectors. arXiv. https://arxiv.org/abs/1207.0580 adresinden 20 Mart 2025 tarihinde alınmıştır.

Hochreiter, S., & Schmidhuber, J. (1997). Long short-term memory. Neural Computation, 9(8), 1735–1780.

Howard, A. G., Zhu, M., Chen, B., Kalenichenko, D., Wang, W., Weyand, T., ... & Adam, H. (2017). Mobilenets: Efficient convolutional neural networks for mobile vision applications. arXiv. https://arxiv.org/abs/1704.04861 adresinden 20 Mart 2025 tarihinde alınmıştır.

Howard, A., Sandler, M., Chu, G., Chen, L. C., Chen, B., Tan, M., ... & Adam, H. (2019). Searching for mobilenetv3. Proceedings of the IEEE/CVF International Conference on Computer Vision (ss. 1314–1324).

Hunter, J. D. (2007). Matplotlib: A 2D graphics environment. Computing in Science & Engineering, 9(3), 90–95. https://doi.org/10.1109/MCSE.2007.55

Iandola, F. N., Han, S., Moskewicz, M. W., Ashraf, K., Dally, W. J., & Keutzer, K. (2016). SqueezeNet: AlexNet-level accuracy with 50x fewer parameters and <0.5 MB model size. arXiv. https://arxiv.org/abs/1602.07360 adresinden 20 Mart 2025 tarihinde alınmıştır.

Jacob, B., Kligys, S., Chen, B., Zhu, M., Tang, M., Howard, A., ... & Adam, H. (2018). Quantization and training of neural networks for efficient integer-arithmetic-only inference. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (ss. 7047–7056).

Krizhevsky, A. (2009). Learning multiple layers of features from tiny images (Technical Report). University of Toronto.

Krizhevsky, A. (2009). CIFAR-10 and CIFAR-100 datasets. https://www.cs.toronto.edu/~kriz/cifar.html adresinden 20 Mart 2025 tarihinde alınmıştır.

Krizhevsky, A., Sutskever, I., & Hinton, G. E. (2012). Imagenet classification with deep convolutional neural networks. Advances in Neural Information Processing Systems, 25, 1097–1105.

Krishnamoorthi, R. (2018). Quantizing deep convolutional networks for efficient inference: A survey. arXiv. https://arxiv.org/abs/1806.08342 adresinden 20 Mart 2025 tarihinde alınmıştır.

Lane, N. D., Georgiev, P., & Qendro, F. (2015). Deep learning at the edge: Opportunities and challenges. Proceedings of the 2nd Workshop on Mobile Systems and Applications (MSA) (ss. 1–5).

LeCun, Y., Bengio, Y., & Hinton, G. (2015). Deep learning. Nature, 521(7553), 436–444.

LeCun, Y., Boser, B., Denker, J. S., Henderson, D., Howard, R. E., Hubbard, W., & Jackel, L. D. (1990). Handwritten digit recognition with a back-propagation network. Advances in Neural Information Processing Systems, 2, 396–404.

LeCun, Y., Bottou, L., Bengio, Y., & Haffner, P. (1998). Gradient-based learning applied to document recognition. Proceedings of the IEEE, 86(11), 2278–2324.

Li, H., Kadambi, A., & Badrinarayanan, V. (2016). Pruning filters for efficient convnets. Workshop Track Proceedings of the International Conference on Learning Representations.

Liu, Z., Yu, C., & Cheng, M. M. (2019). Neural architecture search for deep neural network pruning. Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition (ss. 328–337).

Luo, J. H., Wu, J., & Lin, W. (2017). Thinet: A filter level pruning method for deep neural network compression. Proceedings of the IEEE International Conference on Computer Vision (ss. 5058–5066).

Ma, N., Zhang, X., Zheng, H. T., & Sun, J. (2018). Shufflenet v2: Practical guidelines for efficient CNN architecture design. Proceedings of the European Conference on Computer Vision (ECCV) (ss. 116–131).

McCarthy, J., Minsky, M. L., Rochester, N., & Shannon, C. E. (2006). A proposal for the Dartmouth summer research project on artificial intelligence. AI Magazine, 27(4), 12–14.

Microsoft. (2022). ONNX Runtime Documentation (v1.13.1). https://onnxruntime.ai/ adresinden 20 Mart 2025 tarihinde alınmıştır.

Molchanov, P., Tyree, S., Karras, T., Aila, T., & Kanan, C. (2017). Pruning convolutional neural networks for resource efficient inference. Proceedings of the International Conference on Learning Representations.

Newell, A., & Simon, H. A. (1963). GPS, a program that simulates human thought. In E. A. Feigenbaum & J. Feldman (Eds.), Computers and thought (ss. 279–293). McGraw-Hill.

Newell, A., & Simon, H. A. (1976). Computer science as empirical inquiry: Symbols and search. Communications of the ACM, 19(3), 113–126.

NVIDIA. (2019). Jetson Nano Developer Kit. NVIDIA Developer. https://developer.nvidia.com/embedded/jetson-nano-developer-kit adresinden 20 Mart 2025 tarihinde alınmıştır.

NVIDIA. (2022a). CUDA Toolkit Documentation (v10.2). https://docs.nvidia.com/cuda/ adresinden 20 Mart 2025 tarihinde alınmıştır.

NVIDIA. (2022b). cuDNN Documentation (v8.2.1). https://docs.nvidia.com/deeplearning/cudnn/ adresinden 20 Mart 2025 tarihinde alınmıştır.

NVIDIA. (2022c). TensorRT Documentation (v8.0.1.6). https://developer.nvidia.com/tensorrt adresinden 20 Mart 2025 tarihinde alınmıştır.

Paszke, A., et al. (2019). PyTorch: An imperative style, high-performance deep learning library. Advances in Neural Information Processing Systems, 32, 8026–8037.

Pedregosa, F., et al. (2011). Scikit-learn: Machine learning in Python. Journal of Machine Learning Research, 12, 2825–2830.

PowerTOP Project. (2022). PowerTOP Documentation. Intel. https://01.org/powertop adresinden 20 Mart 2025 tarihinde alınmıştır.

Python Software Foundation. (2022). Python 3.10 Documentation. https://docs.python.org/3.10/ adresinden 20 Mart 2025 tarihinde alınmıştır.

Quinlan, J. R. (1986). Induction of decision trees. Machine Learning, 1(1), 81–106.

Sandler, M., Howard, A., Zhu, M., Zhmoginov, A., & Chen, L. C. (2018). MobileNetV2: Inverted residuals and linear bottlenecks. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (ss. 4510–4520).

Satyanarayanan, M. (2017). The emergence of edge computing. Computer, 50(1), 30–39.

Shortliffe, E. H. (1976). Computer-based medical consultations: MYCIN. Elsevier.

Simonyan, K., & Zisserman, A. (2014). Very deep convolutional networks for large-scale image recognition. arXiv. https://arxiv.org/abs/1409.1556 adresinden 20 Mart 2025 tarihinde alınmıştır.

Szegedy, C., Liu, W., Jia, Y., Sermanet, P., Reed, S., Anguelov, D., ... & Rabinovich, A. (2015). Going deeper with convolutions. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (ss. 1–9).

Tan, M., & Le, Q. V. (2019). EfficientNet: Rethinking model scaling for convolutional neural networks. International Conference on Machine Learning (ss. 6105–6114). PMLR.

TensorFlow Lite. (2022). TensorFlow Lite Guide (v2.9.1). https://www.tensorflow.org/lite/guide adresinden 20 Mart 2025 tarihinde alınmıştır.

TensorFlow Model Optimization Toolkit. (2022). Documentation (v0.7.3). https://www.tensorflow.org/model_optimization adresinden 20 Mart 2025 tarihinde alınmıştır.

TensorFlow-ONNX Contributors. (2022). tf2onnx (Version 1.14.0) [Computer software]. GitHub. https://github.com/onnx/tensorflow-onnx adresinden 20 Mart 2025 tarihinde alınmıştır.

The Pandas Development Team. (2022). pandas-dev/pandas: Pandas (Version 1.5.1). Zenodo. https://doi.org/10.5281/zenodo.7268843

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., ... & Polosukhin, I. (2017). Attention is all you need. Advances in Neural Information Processing Systems, 30, 5998–6008.

Varoquaux, G., Buitinck, L., Louppe, G., Grisel, O., Pedregosa, F., Mueller, A., & Others. (2023). Scikit-learn: Machine learning in Python (Version 1.2.0) [Computer software]. https://scikit-learn.org/ adresinden 20 Mart 2025 tarihinde alınmıştır.

Weizenbaum, J. (1966). ELIZA—A computer program for the study of natural language communication between man and machine. Communications of the ACM, 9(1), 36–45.

Zhang, X., Zhou, X., Lin, M., & Sun, J. (2018). ShuffleNet: An extremely efficient convolutional neural network for mobile devices. Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition (ss. 6848–6856).

Zhu, C., Han, S., Mao, H., & Dally, W. J. (2016). Trained ternary quantization. arXiv. https://arxiv.org/abs/1612.01064 adresinden 20 Mart 2025 tarihinde alınmıştır.

Zhu, M., & Gupta, S. (2017). To prune, or not to prune: Exploring the efficacy of pruning for model compression. arXiv. https://arxiv.org/abs/1710.01878 adresinden 20 Mart 2025 tarihinde alınmıştır.

ÖZGEÇMİŞ

Ad-Soyad : Asım Bilal YILMAZ

ÖĞRENİM DURUMU:

- **Lisans :** 2016, Sakarya Üniversitesi, Bilgisayar ve Bilişim Sistemleri, Bilgisayar Mühendisliği