T.C.

KIRIKKALE ÜNİVERSİTESİ

FEN BİLİMLERİ ENSTİTÜSÜ

BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI

DOKTORA TEZİ

Medikal Görüntülerde İlgi Duyulan Bölge Analizi ve Yeni Paralel Sıkıştırma Yöntemi Geliştirilmesi

Erdal ERDAL

TEMMUZ 2017

Bilgisayar Mühendisliği Anabilim Dalında Erdal ERDAL tarafından hazırlanan MEDİKAL GÖRÜNTÜLERDE İLGİ DUYULAN BÖLGE ANALİZİ VE YENİ PARALEL SIKIŞTIRMA YÖNTEMİ GELİŞTİRİLMESİ adlı Doktora Tezinin Anabilim Dalı standartlarına uygun olduğunu onaylarım.

Prof. Dr. Ali ERİŞEN

Anabilim Dalı Başkanı

Bu tezi okuduğumu ve tezin **Doktora Tezi** olarak bütün gereklilikleri yerine getirdiğini onaylarım.

Yrd. Doç. Dr. Atilla ERGÜZEN

Danışman

Jüri Üyeleri

Başkan	: Prof. Dr. Mehmet Reşit TOLUN	_____
Üye (Danışman)	: Yrd. Doç. Dr. Atilla ERGÜZEN	_____
Üye	: Yrd. Doç. Dr. Abdül Kadir GÖRÜR	_____
Üye	: Yrd. Doç. Dr. Halil Murat ÜNVER	_____
Üye	: Yrd. Doç. Dr. Murat LÜY	_____

……/…../2017

Bu tez ile Kırıkkale Üniversitesi Fen Bilimleri Enstitüsü Yönetim Kurulu Doktora derecesini onaylamıştır.

Prof. Dr. Mustafa YİĞİTOĞLU

Fen Bilimleri Enstitüsü Müdürü

Sevgili Eşime

ve Canım Aileme

ÖZET

Medikal Görüntülerde İlgi Duyulan Bölge Analizi ve Yeni Paralel Sıkıştırma Yöntemi Geliştirilmesi

ERDAL, Erdal
Kırıkkale Üniversitesi
Fen Bilimleri Enstitüsü
Bilgisayar Mühendisliği Anabilim Dalı, Doktora tezi
Danışman: Yrd. Doç. Dr. Atilla ERGÜZEN
Temmuz 2017, 148 sayfa

Dijital medikal görüntüler, sağlık hizmetlerinin her aşamasında yaygın şekilde kullanılmaktadır. Bu medikal görüntüleri aktarmak ve depolamak tıbbi uzmanlar ve hastalar için hayati role sahiptir. Gün geçtikçe artan büyük dosya boyutları ve depolama alanı gereksinimleri nedeniyle görüntü sıkıştırma bir zorunluluk haline gelmiştir. Görüntünün tamamını sıkıştırmak yerine, ilgi alanını (region-of-interest - ROI) sıkıştırmak bu alanda alternatif bir seçenektir. Tüm görüntüye kayıpsız sıkıştırma yöntemleri uygulanması yeterli avantaj sağlamaz, ancak, kayıplı teknikler kullanıldığında da tıbbi görüntünün hayati öneme sahip bilgileri kaybolabilir. Bu tezde ROI ve Optik Karakter Tanıma (Optical Character Recognition - OCR) temelli yeni bir tıbbi görüntü arşivleme sistemi uygulaması önerilmiştir. Ayrıca, daha iyi sıkıştırma oranı ve performansı üretmek üzere özel olarak tasarlanmış yeni bir dinamik dosya yapısı kullanılmıştır. Medikal görüntü ROI ve non-ROI olmayan kısımlara ayrılmıştır. Medikal görüntünün ROI bölümüne kayıpsız sıkıştırma algoritması olan JPEG-LS uygulanmış, OCR ve Huffman kodlama algoritmaları ise görüntünün non-ROI kısmı için kullanılmıştır. Geliştirilen yöntem gerçek hastaya ait beyin MR görüntüleri kullanılarak değerlendirilmiş ve görüntünün non-ROI bölümü için elde edilen sıkıştırma oranı %92,12 ile %97,84 olarak tespit edilmiştir. Önerilen yöntem ile literatürdeki son teknoloji arasındaki ortalama fark ROI dışı kısım için %83,80 tespit edilmiştir. Sonuç olarak, önerilen yöntem medikal görüntü arşivleme problemine

entegre bir çözüm sunmaktadır. Çalışmanın ikinci aşamasında, Hadoop tabanlı MapReduce programlama modelini kullanan bir platform geliştirilmiştir. Platformun performans gereksinimlerini karşılamak için NoSQL veritabanı olan MongoDB kullanılmıştır. Dört düğümlü bir Hadoop kümesi platforma kurulmuştur. Bulut tabanlı çalıştırılan algoritmalar, tek düğümden daha verimli veri işleme yetkinliğine sahiptirler. Platformun performansını doğrulamak için aynı test görüntüleri kullanılmıştır. Test görüntülerinin işlenmesi tek bir düğüm üzerinde 15,599 saniye sürmektedir, ancak bulut tabanlı geliştirilen platformda bu değer 8,153 saniye olarak tespit edilmiştir. Sonuç olarak çalışmanın bu kısmında önerilen yöntem medikal görüntü arşivleme problemine bulut tabanlı bir çözüm sunmaktadır.

Anahtar Kelimeler: Medikal görüntü, ilgi duyulan bölge, görüntü bölütleme, görüntü sıkıştırma, dağıtık paralel sistemler

ABSTRACT

Analysis of Region of Interest (ROI) In Medical Images and a New Compression Method Development

ERDAL, Erdal
Kırıkkale University
Graduate School of Natural and Applied Sciences
Department of Computer Eng., Ph. Degree Thesis
Supervisor: Asst. Prof. Dr. Atilla ERGÜZEN
July 2017, 148 pages

Digital medical images have been widely used in all stages of healthcare. It has a vital role to transfer and to store digital medical images for medical experts and patients. Since the large file sizes and storage space requirements, image compression has become a necessity. Instead of compressing the entire image, it is an option to compress the region of interest (ROI). Applying lossless methods to the whole image does not provide a sufficient advantage, however, when lossy techniques are used, the vital information of the medical image may be lost. In this dissertation, a novel medical image archiving system implementation based on ROI and Optical Character Recognition (OCR) is proposed. Besides, a new dynamic file structure was used that was specially designed to produce better compression ratio and performance. The medical image is separated into the region of interest (ROI) and the non-ROI parts. JPEG-LS, a lossless compression algorithm, is applied to the ROI segment of the medical image. The OCR and Huffman coding algorithm is used for the non-ROI part of the image. The developed method was evaluated using medical images of the actual patient and the produced compression ratio for the non-ROI image is between 92.12% and 97.84%. The average difference between the developed method and the state-of-art in the literature is 83.80% for the non-ROI part. In conclusion, the proposed method provides an integrated solution to the medical image archiving problem. In the second phase of the study, Hadoop developed a platform that uses the MapReduce programming model. MongoDB, a NoSQL database, has been used to satisfy

performance requirements of the platform. A four-node Hadoop cluster has been built to the platform. The algorithms that run on promise more efficient data processing than the single node. The actual patient brain MR images have been used to validate the performance of the platform. The processing of test images takes 15,599 seconds on a single node, but on the developed platform this takes 8,153 seconds. In conclusion of this part of dissertation, the proposed method provides a cloud-based integrated solution to the medical image archiving problem.

Key words: Medical image, region of interest (ROI), image segmentation, image compression, distributed parallel systems

TEŞEKKÜR

Tezimin hazırlanması esnasında hiçbir yardımı esirgemeyen ve büyük destek olan, bilimsel imkanlarını sonuna kadar bizlerin hizmetine sunan, tez yöneticisi değerli hocam, Sayın Yrd. Doç. Dr. Atilla ERGÜZEN'e, tez çalışmalarım esnasında, bilimsel konularda daima yardımını gördüğüm hocam Sayın Prof. Dr. Erdem Kamil YILDIRIM'a, Sayın Prof. Dr. Ertuğrul ÇAM'a, Sayın Yrd. Doç. Dr. Murat LÜY'e, Sayın Yrd. Doç. Dr. Halil Murat ÜNVER'e teşekkür ederim.

Ayrıca eğitim öğretim hayatım boyunca yanımda olan Lisans hocalarım; Prof. Dr. Mehmet Reşit TOLUN'a, Yrd. Doç. Dr. Abdül Kadir GÖRÜR'e, Yüksek Lisans tez danışmanım Yrd. Doç. Dr. Yücel Batu SALMAN'a teşekkür ederim.

Çalışmalarım boyunca yardımlarını esirgemeyen Ahmet ÖZCAN'a, Saliha Gülsen KESKİN'e, Öğr. Görv. Mahmut ÜNVER'e ve Arş.Gör. Hüseyin AYDİLEK'e teşekkür ederim.

Ayrıca hayatım boyunca yanımda olan ve beni her anlamda destekleyen anneme, babama ve kardeşime çok teşekkür ederim. Bu süreçte sabırla yanımda olan ve desteklerini esirgemeyen eşime de çok teşekkür ederim.

İÇİNDEKİLER DİZİNİ

Sayfa

ÖZET.....**i**

ABSTRACT.....**iii**

TEŞEKKÜR.....**v**

İÇİNDEKİLER DİZİNİ.....**vi**

ÇİZELGELER DİZİNİ.....**ix**

ŞEKİLLER DİZİNİ.....**x**

SİMGE VE KISALTMALAR DİZİNİ.....**xiii**

1.GİRİŞ.....**1**

1.1. Medikal Görüntüleme Yöntemleri.....1

1.1.1.Röntgen.....6

1.1.2.Bilgisayarlı Tomografi (BT).....7

1.1.3.Manyetik Rezonans (MR).....9

1.1.4.Ultrasonografi (US).....11

1.2. Görüntü Arşivleme ve İletişim Sistemleri.....12

1.3. DICOM.....14

1.4. Görüntü Sıkıştırma İhtiyacı.....18

1.5. Sıkıştırma Temelleri.....18

1.6. Sıkıştırma Yöntemleri.....19

1.6.1.Kayıpsız Sıkıştırma.....20

1.6.2.Kayıplı Sıkıştırma.....24

1.7. Sıkıştırma Probleminde Kullanılan Yöntemler.....24

1.7.1.Görüntü Bölütleme Yöntemleri.....24

1.7.2.Optik Karakter Tanıma Yöntemi 25
1.7.3.JPEG-LS Algoritması 28
1.7.4.Huffman Kodlama Algoritması 31
1.8. Bulut Tabanlı Sıkıştırma Probleminde Kullanılan Yöntemler 33
1.8.1.Görüntü İşleme Paketi 33
1.8.2.MapReduce/Hadoop 33
1.8.3.MongoDB ve GridFS 36
1.9. Çalışmanın Amacı ve Katkıları 40
2.LİTERATÜR İNCELEMESİ 42
3.MATERYAL VE YÖNTEM 47
3.1. Veri Seti 47
3.2. Geliştirme Ortamı 50
3.3. Sıkıştırma Problem Tanımı 50
3.3.1. Geliştirilen Yöntem 51
3.3.2. Görüntü Parçalama İşlemi 53
3.3.3. ROI Bölgesi İşlemleri 60
3.3.4. ROI Dışı Bölgesi İşlemleri 61
3.3.5. Saklama ve Arşivleme İşlemleri 64
3.4. Bulut Tabanlı Sıkıştırma Problem Tanımı 65
3.4.1. Geliştirilen Yöntem 65
3.4.2. Hadoop Ön İşleme Süreci 66
3.4.3. MapReduce/Hadoop Kullanımı 73
3.4.4. MongoDB ve GridFS Kullanımı 80
3.4.5. Arama Motoru 84
3.4.6. Tersine Önişleme 87
4.BULGULAR 88
4.1. Sıkıştırma Sonuçları 88

4.2. Bulut Tabanlı Sıkıştırma Sonuçları 91
4.2.1. MapReduce/Hadoop Performansı 91
4.2.2. MongoDB Performansı 92
5.SONUÇLAR 95
KAYNAKLAR 97
EKLER 103
EK 1 103
EK 2 109
EK 3 112
EK 4 113
EK 5 126
EK 6 127
EK 7 129
EK 8 134
EK 9 143
EK 10 144
ÖZGEÇMİŞ 147

ÇİZELGELER DİZİNİ

ÇİZELGE **Sayfa**

1.1. Biyomedikal görüntüler ve boyutları 2
1.2. Görüntü sınıfları 3
1.3. Radyografi temel yoğunluklar 4
1.4. Huffman kodlama veri örneği 31
3.1. Test görüntü detayları 50
3.2. Görüntü bölütleme karşılaştırması 53
3.3. OCR işlem sonucu 63
4.1. Karşılaştırılmalı sıkıştırma oranları 88
4.2. Dosya boyutları 89
4.3. Huffman sıkıştırma sonuçları 90
4.4. Veri tabanı karşılaştırma test sorguları 93
4.5. Veri tabanı karşılaştırma test sorguları 93
4.6. Veri tabanı veri yazma kıyaslaması 93
4.7. Veri tabanı indeksli sayısal veri çekme kıyaslaması 94
4.8. Veri tabanı indeksli olmayan veri çekme kıyaslaması 94

ŞEKİLLER DİZİNİ

ŞEKİL **Sayfa**

1.1. Beş ana radyografik yoğunluk 5
1.2. Röntgen filmi 6
1.3. Röntgen cihazı 7
1.4. Bilgisayarlı tomografi cihazı 8
1.5. Bilgisayarlı tomografi görüntüsü 8
1.6. MR cihazı 9
1.7. MR görüntüsü 10
1.8. MR cihaz kesiti 10
1.9. Ultrasonografi cihazı 11
1.10. Ultrasonografi görüntüsü 12
1.11. Genel iletişim modeli 16
1.12. DICOM medya iletişim modeli 17
1.13. Genel kayıpsız kodlama sistemi: kayıpsız kodlama sistemi blok diyagramı 20
1.14. Uzunluk kodlama örneği 22
1.15. OCR sistem bileşenleri 26
1.16. Temel JPEG-LS ‘ye genel bakış 30
1.17. Kaynak alfabe için huffman kod yapımı örneği 32
1.18. Apache hadoop ekosistemi 34
1.19. Hadoop usta/köle mimarisi 35
1.20. GridFS mimarisi 38
1.21. Sharded olmayan bağlantı modeli 39
1.22. Sharded bağlantı modeli 39
3.1. MR görüntü sayıları 47
3.2. MR görüntü boyutu 48
3.3. Test görüntüleri 49
3.4. Geliştirilen yöntem akış şeması 52
3.5. Chan-vese aktif kontur yöntemi 1000 tekrarlanma 54

3.6. Chan-vese aktif kontur yöntemi 2200 tekrarlanma 54
3.7. Chan-vese aktif kontur yöntemi 4000 tekrarlanma 55
3.8. Bölge tabanlı aktif kontur yöntemi 100 tekrarlanma 56
3.9. Bölge tabanlı aktif kontur yöntemi 1000 tekrarlanma 57
3.10. Bölge tabanlı aktif kontur yöntemi 4000 tekrarlanma 58
3.11. Otsu eşikleme yöntemi non-roı görüntüsü 59
3.12. Otsu eşikleme yöntemi roı bölgesi 60
3.13. OCR metin veri tespiti 61
3.14. OCR metin veri tespit aşamaları 62
3.15. Dinamik dosya yapısı 64
3.16. Hadoop optik karakter tanıma başlangıcı 67
3.17. Hadoop optik karakter tanıma sonuç dosyası kayıt görüntüsü 68
3.18. Hadoop optik karakter tanıma sonuç konsol ekranı 68
3.19. Hadoop huffman kodlama algoritması başlangıç 69
3.20. Hadoop huffman dosya görüntüsü 69
3.21. Hadoop huffman sıkıştırma sonuç ekranı 70
3.22. Hadoop huffman işlem sonucu konsol ekranı 71
3.23. Hadoop roı bölgesi kayıpsız sıkıştırma işlemi konsol ekranı 72
3.24. Hadoop roı bölgesi kayıpsız sıkıştırma işlemi sonuç ekranı 73
3.25. Mapreduce/hadoop işlem adımları 74
3.26. Oracle vm virtualbox 75
3.27. Veri düğüm sanal makinaları 76
3.28. Sanal makina konsol raporları 77
3.29. Ana makina bilgi ekranı 78
3.30. Hadoop servis açılışı 79
3.31. Hadoop dosya sistemi 80
3.32. MongoDB bağlantı bekliyor 81
3.33. MongoDB gridfs mimarisi 82
3.34. MongoDB tam dizi listesi 83
3.35. MongoDB dosya detay görüntüsü 83
3.36. MongoDB yığın detay görüntüsü 84
3.37. Arama motoru giriş sayfası 85
3.38. Arama motoru arama ve sonuç sayfası 85

3.39. Arama motoru iş akışı 86
4.1. Sıkıştırma oranları 89
4.2. İşlem hızları 91

SİMGE VE KISALTMALAR DİZİNİ

CT, BT	Computed Axial Tomography
MRI	Magnetic Resonance Imaging
PET	Positron Emission Tomography
MB	Megabayt
DEM	Digital Electronic Microscopy
DCM	Digital Color Microscopy
US	Ultrasonografi
PACS	Picture Archiving Communication Systems
DICOM	Digital Imaging and Communications in Medicine
ROI	Region-of-Interest
Non-ROI	Non Region-of-Interest
OCR	Optical Character Recognition
HDFS	Hadoop Distributed File System
RDBMS	Relational Database Management Systems
DBMS	Database Management Systems
OSM	Organizational Sustainability Modeling
DCT	Ayrık Kosinüs Transformasyonu
RLC	Run-length Coding
VLC	Variable-length Coding
JPEG	Joint Photographic Experts Group

1. GİRİŞ

Medikal görüntüler Roentgen tarafından X-ışınlarının 1895 yılında bulunması ile ortaya çıkmıştır [1]. Gelişen teknoloji ve yapılan çalışmalar sonucunda medikal görüntüleme bugünlere gelmiştir ve gerek teşhis gerek tedavi aşamalarında öncelikli olarak kullanılan bir yöntem olmuştur. Günümüzde yüksek çözünürlüklerde medikal görüntüler alınabilmektedir. Ancak büyük dosya boyutları medikal görüntülerin iletilmesi ve arşivlenmesi gibi önemli alanlarda aşılması gereken problemlere sebep olmuştur.

1.1. Medikal Görüntüleme Yöntemleri

Günümüzde sağlık taraması sürecinde kullanılan medikal görüntü yöntemi ve kalitesi hastalığın teşhis ve tedavi sürecinde etkin rol almaktadır [2]. Medikal görüntüler, sadece hastaların iç organlarının, dokularının, kemiklerinin ve diğer özelliklerinin anatomik enine kesitlerini sağlamakla kalmaz aynı zamanda doktorlar ve tüm sağlık personelleri için hastanın teşhisinin değerlendirilmesinde ve uygulanan tedavinin etkilerinin değerlendirilmesinde kullanılmaktadır. Medikal görüntüler bilgisayarlı tomografi (computed axial tomography – CT), manyetik rezonans görüntüleme (magnetic resonance imaging – MRI), pozitron yayınlayıcı tomografi (positron emission tomography – PET) ve dijital mamografi gibi geniş görüntüleme kaynaklarından sağlanmaktadırlar. Örneğin, 512 x 512 x 16 bit 30 görüntü taraması içeren tek bir CT taramasında oluşan veri boyutu yaklaşık 15 megabayttır (MB). Bir sağlık merkezi için yılda yaklaşık 8 milyon ayrı resim parçası üretildiği tahmin edilmektedir. Bu değer, tipik bir resim boyutunun 0.5 megabayt olduğu düşünüldüğünde, 4 terabayt (TB) görüntü verisi anlamına gelmektedir.

Günlük olarak artan bu medikal görüntülerin depolama ihtiyacı ve veri yönetimi çok büyük boyutlara ulaşmaktadır. Tüm merkezlerde verilerin saklanması, arşivlenmesi ve korunması için güvenilir bir depolama sistemine ihtiyaç duymaktadırlar.

Wang ve arkadaşlarına göre medikal görüntülerde farklı yöntemler, boyutları, gri seviyeli bitleri ve muayene başına ortalama boyutları Çizelge 1.1. Biyomedikal görüntüler ve boyutlarında gösterilmektedir [3].

İki boyutlu tıbbi bir görüntü, M x N x K bit boyutuna sahiptir. Çizelge 1.1. Biyomedikal görüntüler ve boyutlarında görüldüğü üzere, çeşitli görüntüleme teknolojileri tarafından üretilen muayene başına ortalama tahmini MB değerini listeler; burada 12 bitlik bir resim, belleğin 2 baytıyla temsil edilir. Ayrıca DEM ve DCM hariç diğer tüm yöntemler genellikle radyolojik görüntüler olarak sınıflandırılır ve tanı, tedavi ve cerrahi planlama için kullanılır. Radyolojik görüntüler genellikle Doppler ultrason ve sahte renk nükleer tıp görüntüleri hariç, tek renkli olarak temsil edilmektedir [4].

Çizelge 1.1. Biyomedikal görüntüler ve boyutları

Yöntem	Resim Boyutları	Gri Seviyesi (Bit)	Ortalama Boyut
Nükleer tıp	128^2	8 ya da 16	2 MB
MRI	256^2	12	8-20 MB
Ultrason	512^2	8	5- 8 MB
Doppler Ultrason	512^2	24	15-24 MB
DSA	512^2	8	4-10MB
CT	512^2	12	20 MB
Spiral veya helikal BT	512^2	12	40-150 MB
Dijital Elektronik Mikroskopi	512^2	8	Değişken
Dijital Renkli Mikroskopi (DCM)	512^2	24	Değişken
Kalp Kateterizasyonu	512^2 ya da 1024^2	8	500 – 1000 MB
Sayısallaştırılmış X-ışını Filmleri	2048^2	12	8 MB
Bilgisayar Radyolojisi	2048^2	12	8-32 MB
Sayısallaştırılmış Mamogram	4096^2	12	64 MB

Kullanılan görüntülerin vücutla etkileşimi, enerji türü, görüntü teknolojisi, görüntü türü ve görüntü gösterimi gibi kriterlere göre sınıflandırılmaları Çizelge 1.2. Görüntü sınıflarında gösterilmektedir.

Çizelge 1.2. Görüntü sınıfları

	Vücutla etkileşimi	Enerji türü	Görüntü teknolojisi	Görüntü türü	Görüntü gösterimi
Röntgen	Transmisyon	X-ışını	Analog/ Dijital	Projeksiyon	Sabit/Canlı
BT	Transmisyon	X-ışını	Dijital	Kesit	Sabit
MR	Emisyon	RF/Manyetizma	Dijital	Kesit	Sabit
US	Yankı	Ultrases	Analog/ Dijital	Kesit	Canlı

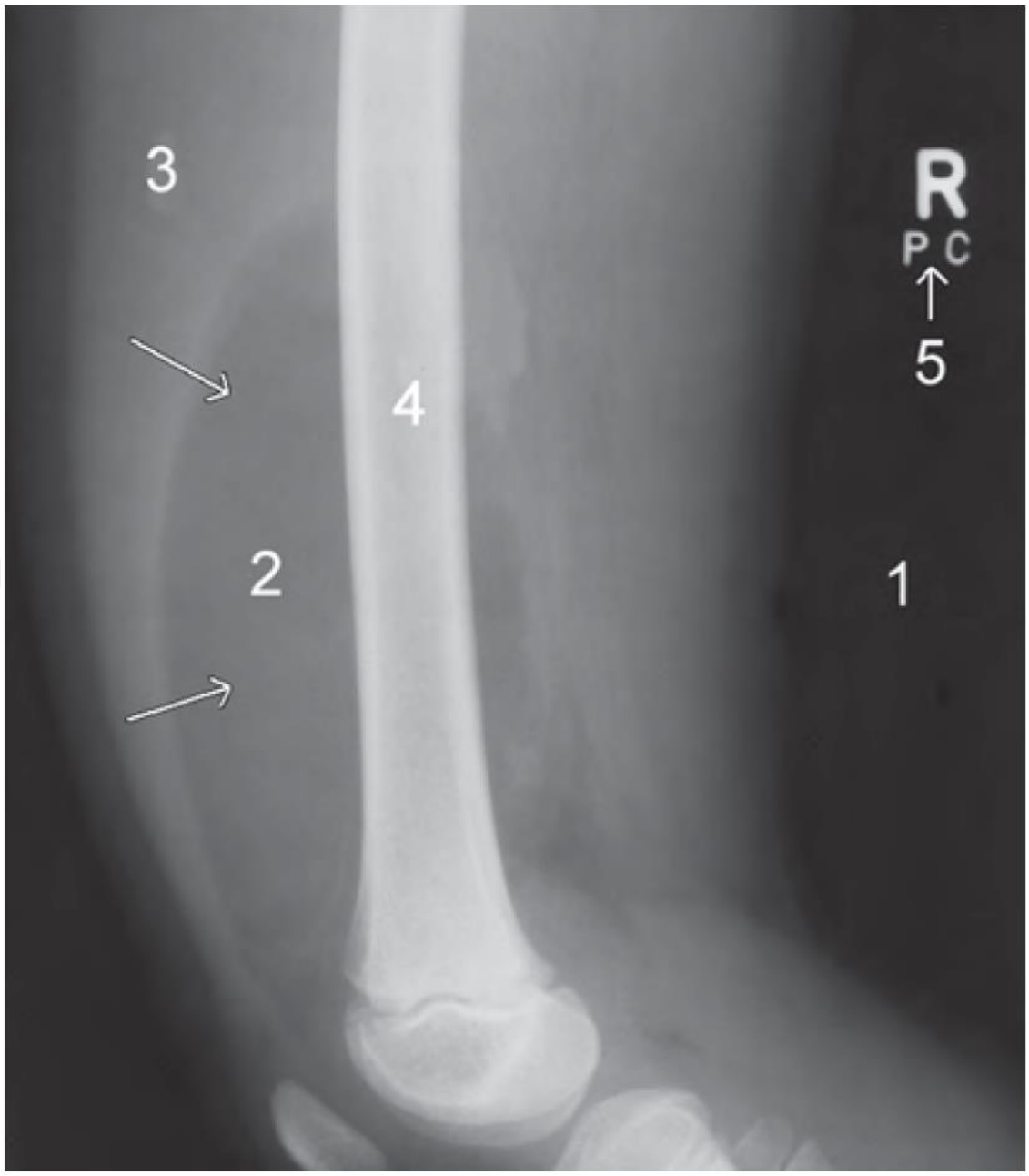
Çizelge 1.2. Görüntü sınıflarında görüldüğü gibi radyografi yani X-ışın tabanlı görüntüleme bu alanda yaygın şekilde kullanılmaktadır. Bu ışınlar, bir volfram elementinden oluşan hedef üzerine, yüksek enerjili bir elektron ışını odaklanarak bir X-ışını tüpü içinde üretilmektedir. X-ışınları, insan vücudundan geçebilen ve iç yapıların bir görüntüsünü üretebilen elektromanyetik radyasyon biçimidir. Oluşan görüntüye, daha yaygın olarak 'X-ışını' veya 'düz film' olarak bilinen radyografi denilmektedir [5].

X-ışınları insan vücudundan geçerken, bazı ışınlar ışının azaltılması veya zayıflatılması ile emilir veya dağılırlar. Yüksek yoğunluklu ve/veya yüksek atom numarası olan dokular, daha fazla X-ışını zayıflatmasına neden olur ve bir radyografda daha açık gri veya beyaz olarak gösterilir. Daha az yoğun dokular ve yapılar, X-ışını demetinin daha az zayıflatılmasına neden olur ve daha yüksek yoğunluklu dokulardan daha koyu görünür. Çizelge 1.3. Radyografi temel yoğunluklarda görüldüğü gibi düz yoğunluk sırasına göre burada sıralanan düz grafilerde beş ana yoğunluk bulunur [5].

Çizelge 1.3. Radyografi temel yoğunluklar

1	Hava / gaz	Siyah	Akciğerler, bağırsak ve karın
2	Yağ	Koyu gri	Subkütan doku tabakası, retroperitoneal yağ
3	Yumuşak dokular / su	Açık Gri	Katı organlar, kalp, kan damarları, mesane gibi kas ve sıvılaştırılmış organlar
4	Kemik	Kapalı Beyaz	
5	Kontrast malzeme/ metal	Parlak beyaz	

Bu yoğunlukları taşıyan örnek bir görüntü Şekil 1.1. Beş ana radyografik yoğunlukta görüldüğü gibidir [5].

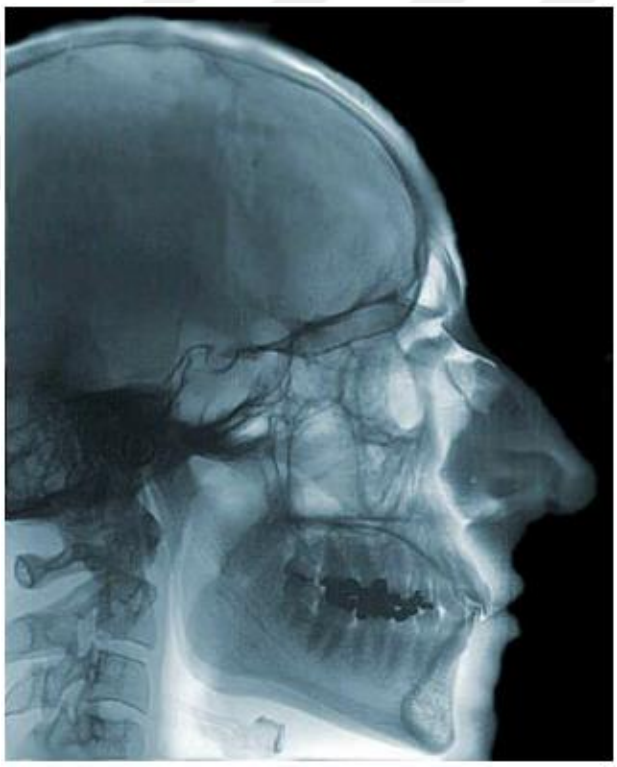


Şekil 1.1. Beş ana radyografik yoğunluk

Yukarıdaki şekilde beş ana radyografik yoğunluk görülmektedir. Şekle göre: (1) hava; (2) yağ; (3) yumuşak doku; (4) kemik; (5) metal kısımlardır.

1.1.1. Röntgen

Röntgenler enerji türü olarak X-ışınlarını kullanmaktadırlar. Bu ışınların kullanılmalarının temel nedeni dokudan yani insan vücudundan geçebilmeleridir. İnsan vücudunun atom ağırlığının, kalınlığının ve yoğunluğunun farklı dokulardan oluştuğu düşünüldüğünde, vücuda verilen X-ışınları farklı emilme ve soğurma gerçekleştirirler. Oluşan farklı emilme/soğurma ve girginlik değerleri ile röntgenogram üzerinde farklı derecelerde düşen ışınlar geçtikleri noktalara ait bir görüntü meydana getirirler. Oluşan bu gri tonlu görüntünün beyaz noktaları ışının en az geçtiği, siyah olan noktalar ise ışının en çok geçtiği noktaları ifade etmektedir [2].



Şekil 1.2. Röntgen filmi

Ancak günümüzde dijital röntgenler kullanılmaktadır. Cihazdan alınan röntgen görüntüleri röntgenogram yerine dijital ortama aktarılır. Bu sayede dijital görüntü

üzerinde oynamalar, renk tonlamalar yapılabileceğinden bu yöntemin avantajları bulunmaktadır.



Şekil 1.3. Röntgen cihazı

Röntgen görüntüler diğer medikal görüntüleme teknolojilerine göre dezavantajlar barındırmaktadır. Öncelikle röntgen filmleri iki boyutlu görüntü sunmaktadırlar. Ayrıca ara yoğunlukta çok fazla görüntü barındırdığından dokular ya da organlar arasında ayrım problemleri yaşanmaktadır.

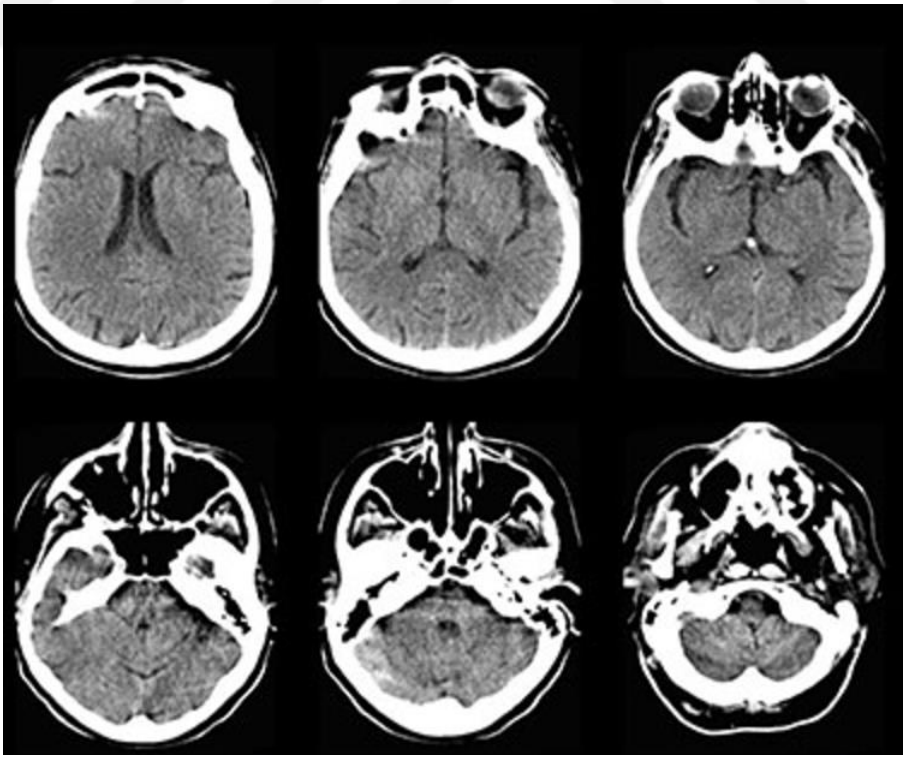
1.1.2. Bilgisayarlı Tomografi (BT)

Tomografi kelimesi Yunan kökenli bir kelime olup kesit görüntü anlamı taşımaktadır [6]. Bilgisayarlı tomografide tıpkı röntgende olduğu gibi X-ışını kullanmaktadır ve sabit şekilde yatırılan hastanın incelenen bölgesinin kesitler halinde görüntüsü alınmaktadır. Hastanın sabitlendiği masa elle ya da otomatik olarak tomografi cihazının "gantry" olarak isimlendirilen açıklığına alınır. Bilgisayar tarafından kontrol edilen bu cihaz X-ışınlarını yönetir ve veriler sayısal veri haline dönüştürülür, bu

sayede BT görüntüleri oluşturulur. Kesitler halinde oluşturulan görüntüler ekranlar aracılığıyla izlenebilir ya da kaydedilebilir.



Şekil 1.4. Bilgisayarlı tomografi cihazı



Şekil 1.5. Bilgisayarlı tomografi görüntüsü

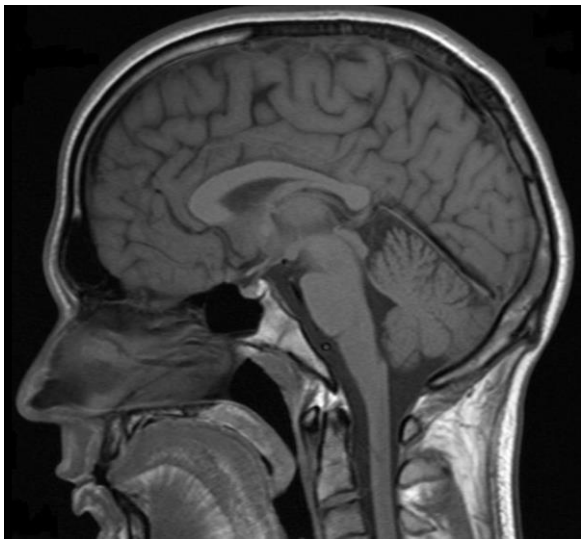
1.1.3. Manyetik Rezonans (MR)

Manyetik rezonans görüntüleme radyasyon içermeyen kesit görüntüler kullanan yumuşak doku kontrastını arttıran bir görüntüleme teknolojisidir. Bu teknoloji arkasında radyo dalgaları ve güçlü bir mıknatıs bulunmaktadır. MR teknolojisi hidrojen çekirdeğini ve hücre sıvısını kullanmaktadır. Bu görüntüleme teknolojisinde X-ışını bulunmadığından diğer yöntemlere göre sağlıklı olmakla beraber daha maliyetli bir yöntemdir.

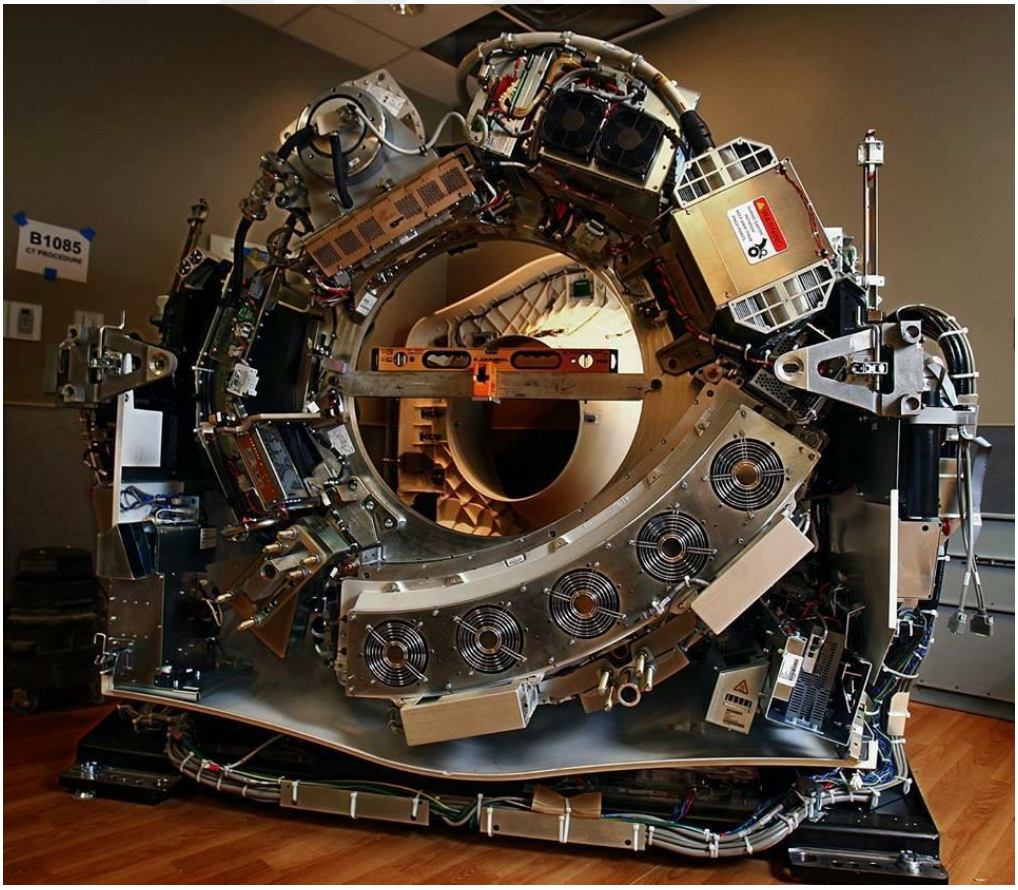
2003 yılında Lauterbur [7] ve Mansfield [8], manyetik rezonans görüntüleme ile ilgili keşifleri sayesinde Nobel Ödülü almışlardır. MR, hastayı zararlı iyonlaştırıcı radyasyona maruz bırakmadığından verimlidir ve yapısal görüntüler için 1 mm'den daha az izotropik çözünürlük sağlayabilir.



Şekil 1.6. MR cihazı



Şekil 1.7. MR görüntüsü



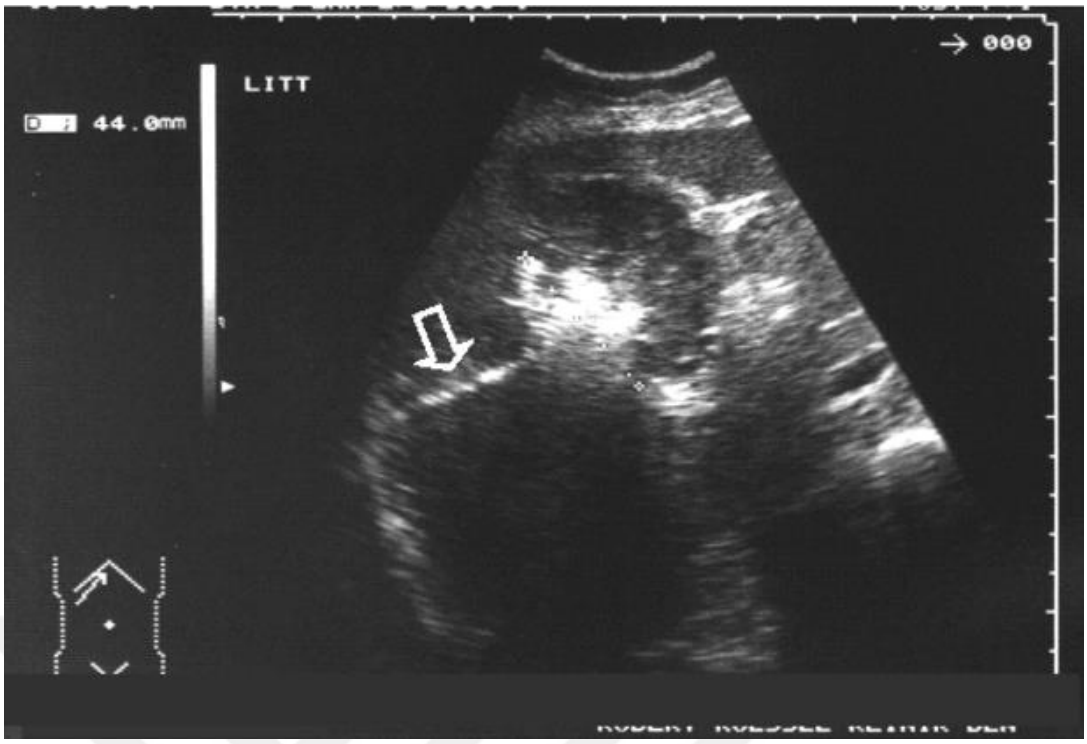
Şekil 1.8. MR cihaz kesiti

1.1.4. Ultrasonografi (US)

Ultrasonografi görüntüleme teknolojisinin altında yüksek frekanstaki ses dalgaları bulunmaktadır. Vücuda yüksek frekanslı ses dalgaları gönderilir ve vücudun farklı noktalarındaki yüzeylerden bu dalgaların yansımaları tespit edilir. Ultrason insan kulağının duyabileceğinden daha yüksek frekansa sahip ses dalgalarından oluşur. Bu yöntemin en büyük avantajı X-ışınlarını kullanmaması yani radyasyona gereksinim duymamasıdır. Bir diğer avantajı ise ultrasonografi çekimi esnasında alınan görüntüler gerçek zamanlı olarak monitörden izlenebilmektedir.



Şekil 1.9. Ultrasonografi cihazı



Şekil 1.10. Ultrasonografi görüntüsü

1.2. Görüntü Arşivleme ve İletişim Sistemleri

Bir uzman muayene odasındaki bir hastayla diyalog halinde iken hastane bilgi sistemi üzerinden radyografik bir muayene talimatı verir. Birkaç dakika içinde bu talep radyograj isteyen hastalardan oluşan bir listede görünür. Radyoloji teknisyenleri uygun hastayı listeden seçer ve çalışmayı yürütür. Görüntüler oluşturulduktan sonra merkezi bir yerel sunucuya gönderilir ve talep otomatik olarak tamamlanır ve hastane bilgi sisteminde faturalandırılır. Görüntüler kullanımı kolay, tarayıcı tabanlı bir tıbbi görüntü izleyicisi veya yerel iş istasyonu programındaki bir dijital görüntüleme ve iletişim programında (DICOM) bilgisayar üzerinden hastane bilgi sistemindeki görüntüler sekmesinin altındaki "görüntüle" düğmesini tıklanarak görülebilir. Bu işlemde senaryoyu görüntüleme, kontrast ve parlaklığın gerçek zamanlı olarak ayarlanmasına, büyütülmesine, ölçümlerin gerçekleştirilmesine ve diğer birçok görüntü izleme özelliğine olanak tanınır. Bir Resim Arşivleme ve İletişim Sistemi'nin (Picture Archive and Communication System - PACS) temel işlevi budur. Hiçbir veri giriş hatası yaşanmaz, görüntüler kâğıt ortamında olmaz ve silinemez. Buna ek olarak,

görüntülere ve raporlara yerel alan ağı (LAN) içindeki herhangi bir yerden erişilebilir ve ayrıca web üzerinden yerel ağı dışında da erişilebilir. Ek olarak, uzmanlar bir uygulamadan dışarı çıkıp başka bir uygulamayı açıp farklı bir uygulama içindeki görüntüleme sınaması için vakit harcamak zorunda kalmazlar [9-11].

PACS, dijital görüntülerin oluşturulması, dağıtılması ve arşivlenmesine katkıda bulunan teknolojileri barındırır [12]. Tipik PACS bileşenleri, bir dijital görüntüleme modalitesi veya cihazı, makul bant genişliği olan bir ağ, bir arşiv cihazı, teşhis iş istasyonları, arşiv/yönlendirme yazılımı ve genellikle hastane veya radyoloji bilgi sistemiyle bazı etkileşimleri içerir. Bilginin belirli bir zaman aralığında iletilebileceği hız olarak tanımlanan bant genişliği önemlidir, çünkü tıbbi görüntü dosyaları büyüktür ve çalışma başına ortalama 10-50 Megabayt (MB) düzeyindedir [12]. Karşılaştırma için, bir dijital fotoğraf makinesiyle yapılmış bir dijital fotoğraf, 250-500 kilobayt (yaklaşık 50-100 kat daha küçük) bir dosya oluşturabilir.

Tüm modern PACS sistemleri DICOM standardını kullanmaktadır. DICOM, mevcut internet standart protokolünün (TCP / IP) üstünde çalışan bir ağ protokolünü kullanır. PACS sistemlerinin kullanılmasının avantajları aşağıdaki gibidir [11]:

- Radyografik film ve kimyanın azaltılmış tüketimi, işlemci iş yükü/bakımının azaltılması, basılı kopya depolama alanı için daha az gereklilik ve filmin taşınması ve dağıtımı ile ilgili işçilik maliyetinin düşürülmesi ile doğrudan maliyet tasarrufu sağlar,
- Tesisler ve bölümler arasında bağlantı ve entegrasyon artar,
- Verimlilik iyileştirmeleri sağlar (yanlış yerleştirilmiş radyograflar veya ultrason resimleri aramak için harcanan zaman, filmi asmak/filme almak için harcanmış vakitleri azaltır, filmleri dağıtmak için daha az zaman harcanmasını sağlar),
- Aynı görüntülerin birden çok yerde aynı anda görüntülenmesini sağlar,
- Kayıp muayeneleri ortadan kaldırarak ve etkin kapasiteyi arttırarak gelirleri arttırır,
- Analog film veya termal baskılardan daha iyi görüntü kalitesi sağlar,
- Tanıları yorumlamak ve iletmek için zamanı azaltır,

- Uzmanlara hızlı danışma için bir yol sağlar.

Tüm bu gelişmelerin ve faydaların yanında sistemde günden güne problemler oluşmaya başlamıştır. Örneğin, bir PACS sistemi 7 yılda 40 terabayt depolama kapasitesine ihtiyaç duyar. Bu örnekten yola çıkarak, yüksek kapasite ihtiyacı, sınırlı uzaysal çözünürlük, yavaş görüntü gösterimi, herhangi bir bileşende oluşacak bir hataya karşı yedekleme ve felaket kurtarma sistemi gereksinimi gibi nedenler PACS sistemlerinin eksiklikleri olarak tanımlanmıştır [9].

PACS, görüntü yakalama cihazları, veri yönetim sistemi, iletim ağı, görüntü gösterim istasyonu ve basılı kopya istasyonu gibi birçok alt sistemden oluşmaktadır.

1.3. DICOM

Amerikan Radyoloji Koleji (American College of Radiology - ACR) ve Ulusal Elektrik Üreticileri Birliği (National Electricity Manufacturers Association - NEMA), aşağıdakilere yönelik bir standart geliştirmek için 1983'te ortak bir komite oluşturmuştur [13]:

- Aygıt üreticisine bakılmaksızın, dijital görüntü bilgilerinin ortak bir platformda iletişimini sağlamak,
- Diğer hastane bilgi sistemleri ile de ara yüz oluşturabilen resim arşivleme ve iletişim sistemlerinin (PACS) geliştirilmesi ve genişletilmesinin kolaylaştırılması,
- Coğrafik olarak dağıtılan çok çeşitli cihazlar tarafından sorgulanabilecek tanısal bilgi veri tabanlarının oluşturulmasına izin vermek.

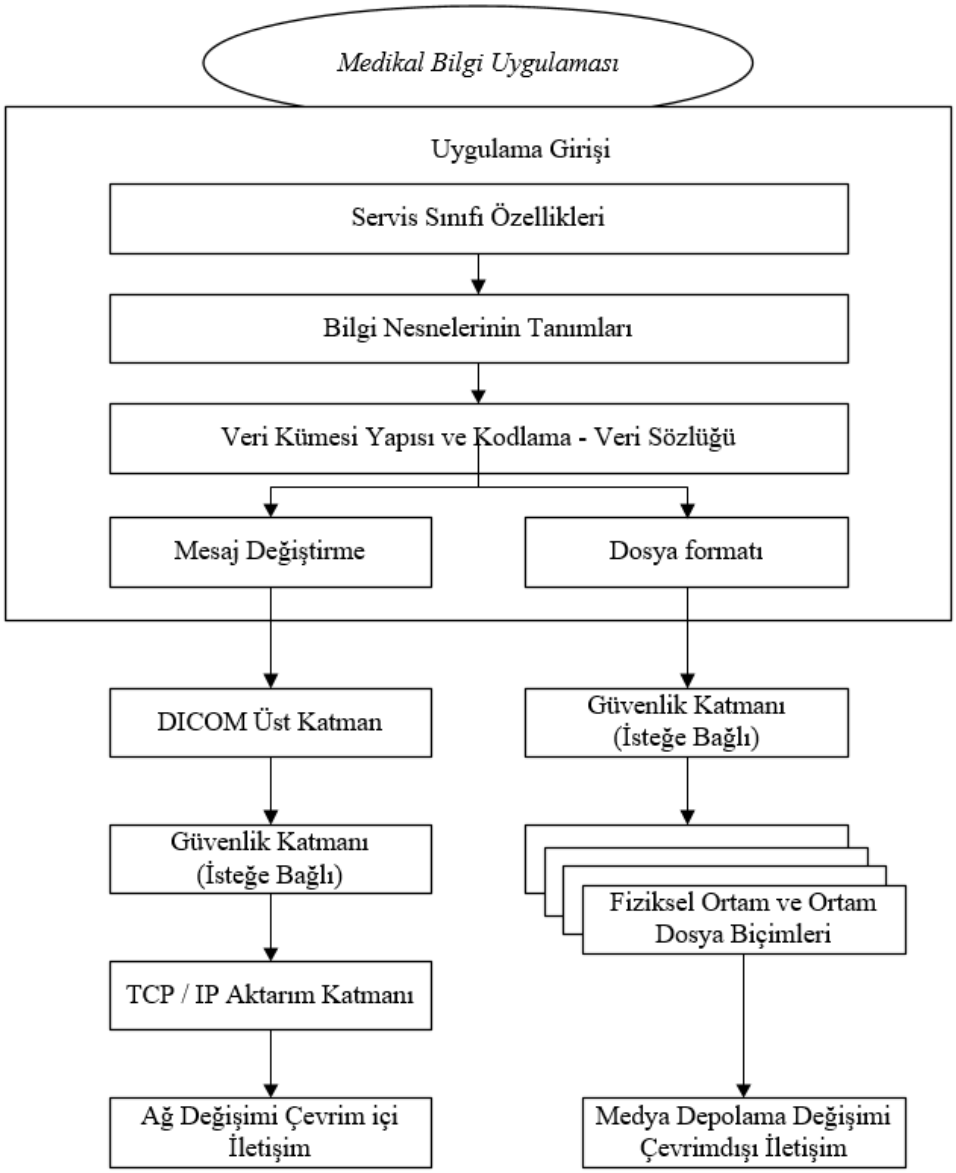
DICOM Standardı, medikal görüntüleme ekipmanının birlikte çalışabilirliğini aşağıdakileri belirterek kolaylaştırır [13]:

- Ağ iletişimleri için standartlara uygunluk talebinde bulunan cihazlar tarafından takip edilecek protokoller kümesi oluşturmak.
- Komutların söz dizimi ve semantiği ve bu protokolleri kullanarak değiştirilebilecek ilişkili bilgileri sağlamak.

DICOM standardı, uyumluluk iddia eden cihazların birlikte çalışabilirliğini kolaylaştırır. Özellikle aşağıdaki maddelerle bu durum sağlanır [14]:

- Komutların ve ilişkili verilerin anlambilimlerini ele alır. Cihazların etkileşimde bulunabilmesi için cihazlar arasında taşınması gereken bilgileri değil, cihazların ve ilgili verilere tepki vermesinin nasıl olacağı konusunda standartlar geliştirmektir.
- Çevrimdışı iletişim için gerekli dosya servisleri, dosya formatları ve bilgi dizinlerinin semantiklerini ele alır.
- Ağ üzerinde çalışmayı kolaylaştırır.
- Yeni servislerin kullanıma sunulması için yapılandırılmıştır, böylece gelecek tıbbi görüntüleme uygulamalarına zemin hazırlar.
- Mevcut uluslararası standartların uygulanabilir olduğu yerlerde kullanılmasını sağlar ve kendisi uluslararası standartlar için belirlenmiş dokümantasyon kurallarına uymaktadır.

Bu standart, radyoloji, kardiyoloji ve ilgili disiplinlerde uygulanan teşhis tıbbi görüntülemeye vurgu ile geliştirilmiştir; Bununla birlikte, klinik ve diğer tıbbi ortamlarda değiş tokuş edilen geniş bir yelpazede imge ve resimle ilgili olmayan bilgiler için de geçerlidir. Şekil 1.11. Genel iletişim modelinde hem ağ (çevrimiçi) hem de medya depolama değişim (çevrimdışı) iletişimini kapsayan standardın genel iletişim modeli gösterilmektedir [13].

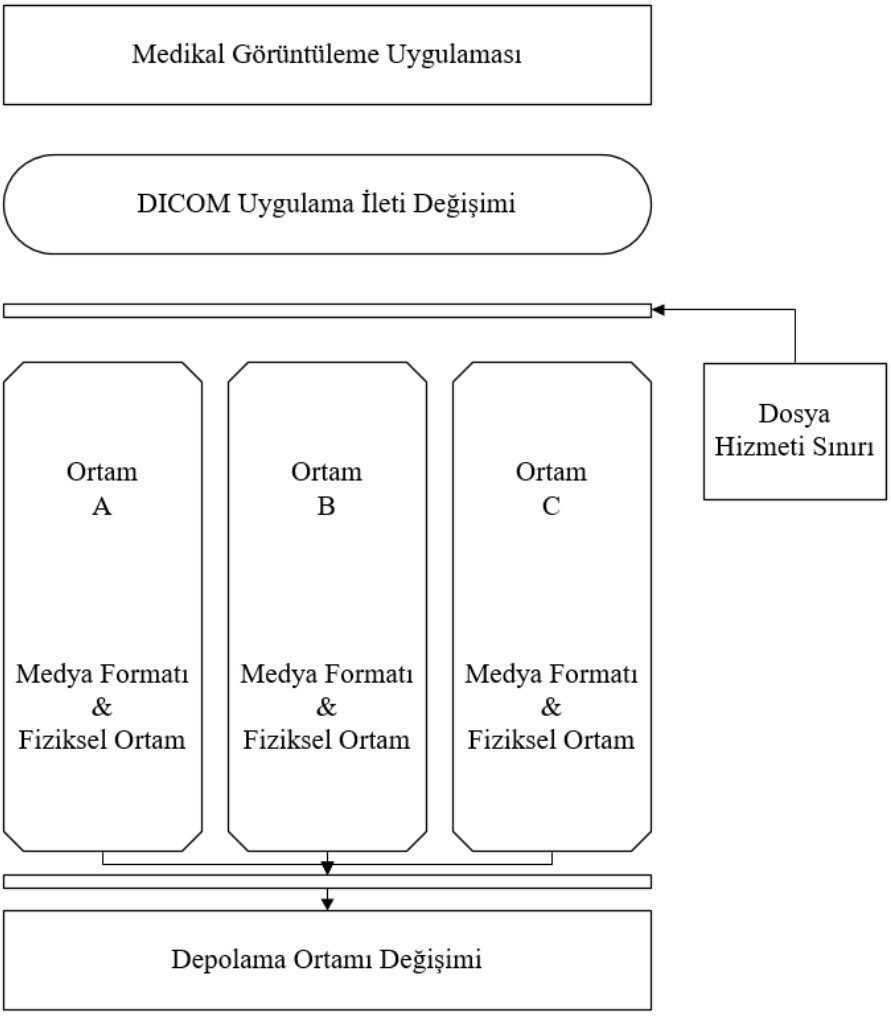


Şekil 1.11. Genel iletişim modeli

Uygulamalar aşağıdaki sınırların herhangi birinde geçiş gösterebilir:

- TCP/IP gibi belirli fiziksel ağ iletişimi desteği ve protokollerinden bağımsız olarak üst katman hizmeti.
- Belirli medya depolama biçimlerinden ve dosya yapılarından bağımsız olarak Depolama Ortamına erişim sağlayan temel DICOM dosya hizmeti.

DICOM standardı, medikal görüntüleme bilgisinin çıkarılabilir ortamda saklanması için genel bir modeli belirtir, bu model Şekil 1.12. DICOM medya iletişim modelinde görülmektedir [13].



Şekil 1.12. DICOM medya iletişim modeli

Tüm bunlara rağmen, DICOM standardı, PACS çözümlerinin uygulanmasını kolaylaştırma potansiyeline sahip olsa da standardı tek başına kullanmak, bir PACS'in tüm hedeflerine uyacağını garanti etmez. Bu standart, çok satıcılı bir ortamda uyumluluk iddia eden sistemlerin birlikte çalışabilirliğini kolaylaştırır, ancak kendi başına birlikte çalışabilirliği garanti etmez [13].

DICOM temelde bir formattır. Bu formatın kullanım amacı MRI, ultrason, CT gibi tüm görüntülerin saklanması, görüntülenmesi ve iletilmesidir [15]. DICOM formatı içerisinde hem medikal görüntüyü hem de hasta tanımlayıcı (meta) bilgileri de bulunmaktadır [16].

1.4. Görüntü Sıkıştırma İhtiyacı

Gelişen teknoloji ile beraber medikal görüntü boyutları da artmıştır. Bu görüntünün uzun yıllar saklanması, zaman içerisinde farklı sistemlere aktarılması gibi nedenlerle ihtiyaç duyulan kapasite terabaytlar seviyesine ulaşmakta ve günden güne artmaktadır. Kayıpsız görüntü sıkıştırmasının amacı, herhangi bir bilgiyi kaybetmeden mümkün olan en küçük bit sayısına sahip bir görüntü sinyalini temsil etmektir; böylece iletim hızlandırılmıştır ve depolama gereksinimlerinin en aza indirgenmesi sağlanmıştır [17]. Bu nedenle elde edilen hasta verilerine (görüntü, ses, video) sıkıştırma yöntemleri uygulanmaktadır. Hasta hayatı söz konusu olduğundan saklanacak tüm bilgilerin en az veri ve detay kaybıyla saklanması gerekmektedir.

Sonuç olarak, verilerin uzun yıllar saklanması ve istenildiğinde farklı lokasyonlara hızlı şekilde ve en az bant genişliği ile aktarılmaları için görüntü sıkıştırma ihtiyacı bulunmaktadır. Bu alan literatürde sıkça çalışılan bir alan haline gelmiştir.

1.5. Sıkıştırma Temelleri

Görüntü sıkıştırma kavramı temelde ihtiyaç duyulmayan ya da gereksiz bilgilerin ayrılması olarak ifade edilebilir ve dosya boyutunun azaltılması amacını taşır. Bu şekilde tanımlanan bilgiler ikiye ayrılır [18].

1- Fazlalık Azaltma

Bu yaklaşımda görüntü içerisinde yer alan ve kendisini tekrar eden fazlalıklar kullanılmaktadır.

a. Uzaysal Fazlalık

Birbirine komşu piksel tabanlı sıkıştırma sağlamaktadır. Genelde görüntülerde bulunan piksel özelliklerinde keskin değişiklikler olmaz ve hatta aynı değerleri taşıyabilirler. Bu yaklaşımda birbirine komşu ve ilişkili pikseller azaltılır.

b. Spektral Fazlalık

Sıkıştırılacak görüntünün farklı renk değerlerinin benzerliklerinden faydalanmaktadır. Bu yaklaşım görüntülerin histogram değerlerini kullanmaktadır.

c. Anlık Fazlalık

Temelde uzaysal fazlalık yaklaşımına sahiptir ancak hareketli görüntülerde kullanılmaktadır ve hareketli görüntüleri oluşturan görüntü çerçevelerinin aralarında bulunan ilişkiyi kullanmaktadır. Arka arkaya gelen çerçeveler hareketli bölüm hariç çok farklılık barındırmaz. Sıkıştırma birbirlerinin aynı olan bu bilgileri azaltır.

2- İlişkisizlik Azaltma

Bu yaklaşım, depolamada gerekli bitleri düşürmek için bilerek kaldırılan gerçek bilgileri kullanır. Fakat bu kaldırma maliyetlidir; kaybolan bilgiler telafi edilemez. Bu nedenle bu yaklaşım kayıplı ve kayıpsız olarak ikiye ayrılır. Veri kaybına tahammülü olmayan sistemlerde kayıpsız sıkıştırma yöntemlerini kullanır, veri kaybına toleransı olan sistemlerde ise kayıplı sıkıştırma yöntemleri kullanılır.

1.6. Sıkıştırma Yöntemleri

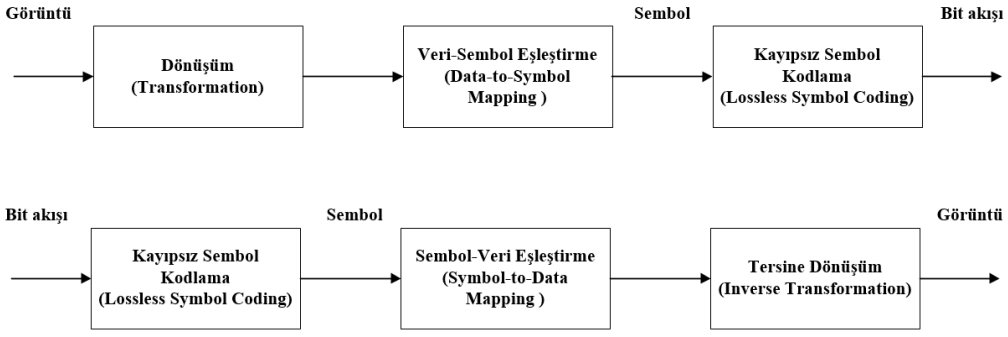
Görüntü sıkıştırma, görüntü işleme konusunda üzerinde sık araştırma yapılan konulardan biri olmakla birlikte, resim boyutunu azaltmak için çok çeşitli yöntemlere sahiptir. Veri sıkıştırma algoritmalarının iki genel kategorisi vardır: kayıpsız ve kayıplı. Kayıpsız sıkıştırma algoritmaları sıkıştırma ve tersine çevirme işlemlerinden sonra girdi dosyasının aynısını oluşturmayı garanti eder [19]. Kayıpsız sıkıştırma algoritmaları arasında Huffman kodlaması, Aritmetik kodlama, Ziv-Lempel kodlaması, çalışma uzunluğu kodlaması, kayıpsız öngörücü kodlama ve bit düzlemi kodlaması bulunur. Kayıpsız sıkıştırma yönteminin aksine, kayıplı sıkıştırma algoritmaları, daha fazla sıkıştırma oranı için orijinal girdi dosyasının doğruluğunu kasıtlı olarak feda eder [19].

Bir veriye uygulanacak sıkıştırma yönteminin kayıplı ya da kayıpsız olmasına yeniden yapılandırma aşamasında veri kaybının oluşup oluşmayacağına bakılarak karar verilebilir [20]. Ayrıca geliştirilen her sıkıştırma yöntemi her veri tipinde aynı sonuçları vermeyebilir, hangi sıkıştırma yönteminin uygulanacağını etkileyen bir diğer faktör ise sıkıştırılmak istenilen verinin tipidir [20].

1.6.1. Kayıpsız Sıkıştırma

Kayıpsız sıkıştırma teknikleri, isminden de anlaşılacağı üzere, bilgide herhangi bir azalmaya ya da kayba neden olmaz. Verilerin sıkıştırılmasında kayıpsız sıkıştırma kullanılmışsa, orijinal veriler herhangi bir kayıp olmaksızın sıkıştırılmış verilerden tam olarak kurtarılabilir. Kayıpsız kodlamada, şifresi çözülmüş görüntü verileri orijinal kodlanmış görüntüye nicel olarak (sayısal olarak) ve nitel olarak (görsel olarak) özdeş olmalıdır [17]. Kayıpsız sıkıştırma, genellikle orijinal ve yeniden yapılandırılmış veriler arasındaki herhangi bir farkı tolere edemeyen uygulamalar için kullanılır [20]. Literatürde birçok kayıpsız sıkıştırma metodu bulunmaktadır [20].

Kayıpsız bir kodlama sisteminin kodlama (encoder) ve çözme (decoder) blok diyagramları Şekil 1.13. Genel kayıpsız kodlama sistemi: Kayıpsız kodlama sistemi blok diyagramında gösterilmektedir [17].



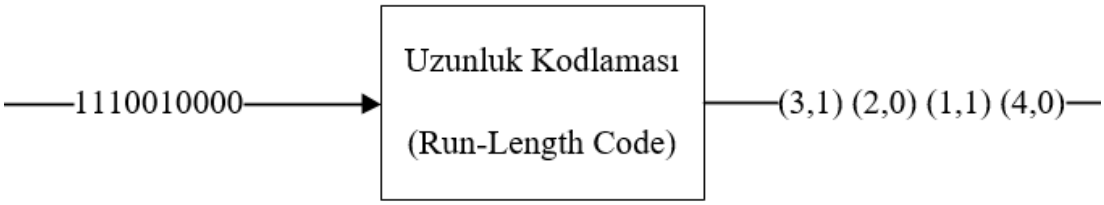
Şekil 1.13. Genel kayıpsız kodlama sistemi: Kayıpsız kodlama sistemi blok diyagramı

Orijinal resmin kodlama aşamasında öncelikli adım dönüşüm adımıdır. Bu aşamada amaç, görüntüyü daha efektif sıkıştırılabilen forma dönüştürmektir. Bu amaçla, seçilen dönüşüm, veri ilişkilendirmesinin azaltılmasına (karşılıklı bağımlılık, fazlalık) yardımcı olabilir, veri istatistiksel dağılımını değiştirebilir veya çok sayıda bilgiyi birkaç veri örneğine veya alt bölge alanlarına paketleyebilir. Tipik dönüşümler, diferansiyel veya prediktif haritalama, ayrık kosinüs transformasyonu (DCT) gibi üniter dönüşümler, dalgacık dönüşümü gibi alt-bant ayrıştırmaları ve yüksek derecede korelasyonu RGB gösteriminden daha az korelasyonu parlaklık-renk değişimine dönüştürme gibi renk uzayı dönüşümlerini içerir [17]. Bu safhada, bu dönüşümlerin bir kombinasyonu da kullanılabilir.

İkinci aşama olan veri-sembol eşleştirme, görüntüleri son aşamada daha efektif kodlayabilmek için sembollere dönüştürür. Bu aşamadaki işlemler örneğin bölümleme ya da uzunluk kodlaması (run-length coding – RLC) ile yapılabilir [17].

Görüntü verileri komşu veri örneklerini bir araya getirerek bloklara bölünebilir; bu durumda her veri bloğu bir semboldür. Birkaç veri birimini bir araya getirmek, görüntü verileri arasında mevcut olabilecek herhangi bir korelasyonun kullanılmasını sağlar ve kodlama karmaşıklığını arttırma pahasına daha yüksek sıkıştırma oranları ile sonuçlanabilir. Buna karşılık, her bir ayrı veri birimi, daha fazla gruplama veya bölme olmaksızın bir sembol olarak alınabilir [17].

RLC'nin arkasındaki temel fikir, bir diziyi sayı-simge çiftleri dizisine (çalışma, değer) eşleştirmektir; burada değer, giriş veri dizisinde bir veri örneğinin değeridir ve veri örneği sürekli olarak tekrarlanır. Bu durumda, her çift (çalışma, değer) bir simgedir. Örnek bir ikili sıralama Şekil 1.14. Uzunluk kodlama örneğinde görülmektedir. Farklı uygulamalar farklı formatlar kullanabilir [17].



Şekil 1.14. Uzunluk kodlama örneği

Kodlamayı optimize etmek için girdi verilerini sembollere uygun şekilde haritalamak çok önemlidir. Uygun veri-sembol haritalama, dikkate alınan uygulamaya ve donanım yazılımının karmaşıklığındaki kısıtlamalara bağlıdır [17].

Kayıpsız sıkıştırmanın son aşaması ise kayıpsız sembol kodlama aşamasıdır. Bu aşama, giriş sembollerine ikili kod kelimeleri atayarak bir ikili bit akışı üretir. Kayıpsız sembol kodlaması, genelde gürültülü kodlama veya sadece kayıpsız kodlama olarak adlandırılır, çünkü bu aşamada, son sıkıştırılmış bit akışına gerçek kayıpsız kodlama uygulanır. İlk iki aşama, verilerin bu kayıpsız kodlama aşamasında daha verimli kodlanabilen bir biçime dönüştürülmesi için önişleme aşamaları olarak kabul edilebilir.

Kayıpsız sıkıştırma genellikle değişken uzunluklu kod sözcükleri kullanılarak yapılır. Burada daha kısa kod sözcükleri daha sık ortaya çıkan simgelere atanır. Bu değişken uzunluklu kod sözcüğü ataması, değişken uzunluklu kodlama (variable-length coding-VLC) ve ayrıca entropi kodlaması olarak bilinir. Huffman ve aritmetik kodlayıcılar gibi entropi kodlayıcıları, sembol oluşmasının olasılığına dayanarak bir dizi sembolü temsil etmek için gereken ortalama bit oranını (simge başına ortalama bit sayısı) en aza indirme girişiminde bulunur. Sıkıştırmayı sağlamak için alternatif bir yol, sabit uzunluklu ikili kod sözcüklerini kullanarak değişken uzunluklu simgeler dizelerini kodlamaktır. Sözlük (Lempel-Ziv) kodlarının ardındaki temel strateji budur [17].

Oluşturulan kayıpsız kod (bit akışı) benzersiz bir şekilde çözülebilmelidir. Diğer bir deyişle, bit akışı belirsizlik olmaksızın şifre çözülebilmekte, sadece bir benzersiz sembol dizisi ortaya çıkmaktadır. Hangi kayıpsız kodlama yönteminin kullanılacağını seçmek, uygulamaya bağlıdır ve uygulama donanımı veya yazılımı, izin verilen kodlama gecikmesi ve gerekli sıkıştırma seviyesi dahil olmak üzere birkaç faktör arasında bir denge gerektirir. Kayıpsız sıkıştırma planı seçerken veya tasarlarken dikkate alınması gereken bazı faktörler aşağıda listelenmiştir.

- Sıkıştırma verimliliği: Orijinal girdi görüntüsünün bit cinsinden toplam boyutunun sıkıştırılmış bit akımı, biti toplam boyutuna bölünmesiyle elde edilir.

- Kodlama gecikmesi: Kodlama gecikmesi hem bir girdi veri örneğini kodlamak hem de çözmek için gereken minimum süre olarak tanımlanabilir. Kodlama gecikmesi toplam aritmetik işlem sayısıyla birlikte artar. Bellek kullanımı genellikle iletişim gecikmelerine yol açtığından, bellek gereksinimlerinde bir artış ile birlikte genellikle artar. Gerçek zamanlı uygulamalar için kodlama gecikmelerinin en aza indirilmesi özellikle önemlidir.

- Uygulama karmaşıklığı: Uygulama karmaşıklığı, gerekli aritmetik işlemlerin toplam sayısı ve bellek gereksinimleri açısından ölçülür. Alternatif olarak, uygulama karmaşıklığı, saniyede gerekli aritmetik işlem sayısı ve belirli bir kodlama gecikmesine veya gerçek zamanlı performans elde etmek için bellek gereksinimleri açısından ölçülebilir. Veritabanı tarama ve alma gibi bazı uygulamalar için, şifreleme, şifre çözme kadar sık yapılmadığından, yalnızca düşük bir şifre çözme karmaşıklığına ihtiyaç duyulmaktadır.

- Sağlamlık: Sıkıştırılmış bit akışının hata eğilimli ortamlarda iletilmesini gerektiren uygulamalar için, kodlama yönteminin iletim hatalarına dayanıklılığı önemli bir husustur.

- Ölçeklenebilirlik: Ölçeklenebilir kodlayıcılar, girilen görüntü verilerinin hiyerarşik bir gösterimini gömerek katmanlı bir bit akışı üretirler. Bu şekilde

giriş verileri hiyerarşik olarak farklı çözünürlüklerde (ölçeklenebilirlik olarak) elde edilebilir ve bit hızı aynı kodlanmış bit akışını kullanan mevcut kaynaklara göre değişebilir (bit hızında ölçeklenebilirlik; kodlama farklı bit hızlarını oluşturmak için tekrarlanmalıdır).

1.6.2. Kayıplı Sıkıştırma

Kayıplı sıkıştırma teknikleri bazı bilgilerin kaybına neden olur ve sıkıştırılmış veri genellikle kurtarılamaz ya da tam olarak yeniden oluşturulamaz [20]. Bir miktar veri kaybının kabullenildiği durumlarda ise daha yüksek sıkıştırma oranları elde edilir. Birçok uygulamada, bu tam yeniden yapılanma eksikliği bir problem değildir [20].

Bu tezde kayıplı sıkıştırma algoritması kullanılmamıştır, bu nedenle kapsam dışı bırakılmıştır.

1.7. Sıkıştırma Probleminde Kullanılan Yöntemler

Bu başlıkta görüntü bölütleme, optik karakter tanıma, JPEG-LS ve Huffman kodlama algoritması gibi bu çalışmada kullanılan yöntemler sunulmaktadır. Bu kısımda yapılan çalışma tek düğüm üzerinde uygulanan bir yaklaşımdır.

1.7.1. Görüntü Bölütleme Yöntemleri

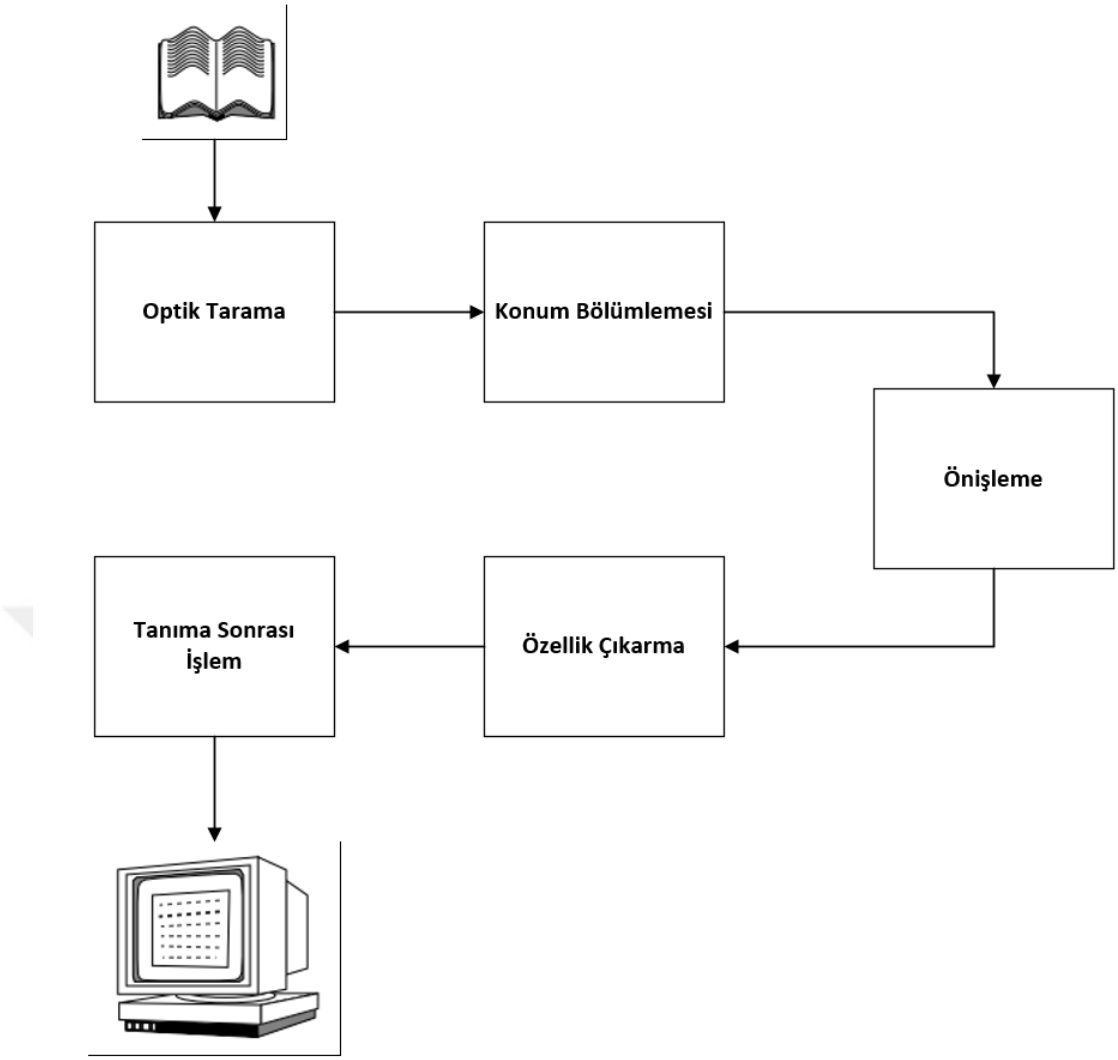
Görüntü bölütleme kavramı, görüntünün talep edilen anlamlı bölgelerine ayrılması olarak tanımlanabilir. Görüntü bölütleme için geliştirilmiş birçok yaklaşım ve model bulunmaktadır. Perona-malik modeli, mumford ve shah yaklaşımı, ambrosio-tortorelli modeli, Chan Vese modeli ve yılan: aktif çevre modeli bunlardan bazılarıdır [21, 22]. Özmen'e göre en hızlı çalışan model yılan: aktif çevre modelidir [21]. Ayrıca Zuo ve arkadaşlarına göre bu modelle medikal görüntüler üzerinde Chan-Vese'ye göre daha başarılı sonuçlar elde edilmiştir [23].

Bu yöntemin ana fikri, enerji minimizasyonu vasıtasıyla bir eğri geliştirmektir. Geliştirildiği tarihten bu yana çok fazla uygulamada kullanılmıştır [21]. Aktif kontur, bir eğri veya bir görüntü içindeki bir yüzey olarak tanımlanır ve arzulanan bir nesnenin veya objenin sınırlarını tespit eder. Algoritma tanımlanan konumdan başlar ve en yakın nesne sınırına doğru hareket eder; bu yüzden "yılan" olarak adlandırılır. Aktif çevre modeli algoritmasında gürültü azaltma özelliği bulunmaz. Bu çalışmada bu özellik medikal görüntüler gibi hayati öneme sahip görüntülerde kullanılmamıştır [22].

1.7.2. Optik Karakter Tanıma Yöntemi

Optik Karakter Tanıma (Optical Character Recognition - OCR), makineleri 1950'lerde piyasaya sürülmüş olmasına rağmen, dünya çapında 1986'ya kadar yalnızca birkaç bin sistem satılmıştı. Bunun temel nedeni, sistemlerin maliyetiydi. Bununla birlikte, donanım daha ucuz hale geldi ve OCR sistemleri yazılım paketleri olarak piyasaya çıkmaya başladıkça, satışlar önemli ölçüde artmıştır. Bugün her hafta satılan sistemlerin sayısı birkaç bindir [24].

Desenlerin otomatik olarak tanınmasında temel prensip, önce makineye hangi desenlerin oluşabileceğini ve neye benzediğini öğretmektir. OCR'de kalıplar harfler, rakamlar ve virgül, soru işaretleri gibi bazı özel simgelerdir. Tipik bir OCR sistemi Şekil 1.15. OCR sistem bileşenlerinde görülen birkaç bileşenden oluşur. İşlemin ilk adımı analog belgeyi bir optik tarayıcı kullanarak dijital hale getirmektir. Metin içeren bölgeler belirlendiğinde, her sembol bir bölümlendirme işlemi yoluyla çıkarılır. Çıkarılan semboller daha sonra bir sonraki aşamadaki özelliklerin çıkarılmasını kolaylaştırmak için gürültüyü ortadan kaldırarak ön işleme tabi tutulabilir [24]. Her bir sembolün kimliği, çıkarılan özellikleri bir önceki öğrenme aşamasında elde edilen sembol sınıflarının tanımlarıyla karşılaştırılarak bulunur. Nihayet içeriğe dayalı bilgiler orijinal metnin sözcüklerini ve sayılarını yeniden oluşturmak için kullanılır.



Şekil 1.15. OCR sistem bileşenleri

İlk işlem optik tarama işlemidir. Tarama işlemi boyunca orijinal belgenin dijital bir görüntüsü çekilir. OCR'de optik tarayıcılar kullanılır; bunlar genelde bir taşıma mekanizması ve ışık yoğunluğunu gri seviyelere çeviren bir algılama cihazı içermektedir. Basılı belgeler genellikle beyaz zemin üzerine siyah baskıyla oluşturulmuştur. Dolayısıyla, OCR gerçekleştirirken, çok düzeyli görüntüyü siyah-beyaz iki katmanlı bir görüntüye dönüştürmek yaygın bir uygulamadır. Eşik olarak bilinen bu işlem genellikle tarayıcıda bellek alanını ve hesaplama çabasını azaltmak için gerçekleştirilir [24].

İkinci işlem konum bölümlemesidir. Bölümleme, bir görüntünün bileşenlerini belirleyen bir işlemdir. Belgenin, verilerin basıldığı bölgelerini bulmak ve onları şekil ve grafiklerden ayırmak gerektiğinden uygulanmaktadır. Metne uygulanan bölümleme, karakterlerin veya kelimelerin izole edilmesidir [24]. Optik karakter tanıma algoritmalarının büyük kısmı sözcükleri tek tek tanıyan izole karakterlere ayırır. Genellikle bu bölümleme, bağlı her bir bileşeni, yani birbirine bağlı siyah alanı izole ederek gerçekleştirilir. Bu tekniğin uygulanması kolaydır, ancak karakterler dokunursa veya karakterler parçalanırsa veya birkaç parçadan oluşursa sorunlar ortaya çıkar.

Sonraki işlem önişleme sürecidir. Tarama işleminden kaynaklanan görüntü belirli bir miktarda gürültü içerebilir. Tarayıcı üzerindeki çözünürlüğe ve eşik değer için uygulanan tekniğin başarısına bağlı olarak, karakterler bulaşabilir veya kırılabilir. Daha sonra zayıf tanıma oranlarına neden olabilecek bu kusurlardan bazıları, pürüzsüzleştirmek için bir ön işlemci kullanarak ortadan kaldırılabilir. Düzeltme hem doldurma hem de inceltme anlamına gelmektedir. Doldurma, sayısallaştırılmış karakterlerdeki küçük kırılmaları, boşlukları ve delikleri ortadan kaldırırken, inceltme çizginin genişliğini azaltmaktadır. Düzleştirmenin en yaygın teknikleri, pencerenin içeriğine belirli kurallar uygulayarak bir pencereyi karakterin ikili görüntüsüne taşımaktadır [24].

Sonrasında özellik çıkartma işlemi uygulanır. Özellik çıkarımının amacı, sembollerin temel özelliklerini yakalamaktır ve genellikle bunun kalıp tanımanın en zor sorunlarından biri olduğu kabul edilmektedir. Bir karakteri tanımlamanın en doğru yolu, gerçek raster görüntüdür. Bir diğer yaklaşım ise, sembolleri karakterize eden bazı özelliklerin çıkarılması ancak önemsiz niteliklerin dışına çıkmasıdır.

Son işleme ise son adımdır. Bir belgede düz sembol tanımanın sonucu, bireysel sembollerin bir kümesidir. Bununla birlikte, bu sembollerin kendileri genellikle yeterli bilgi içermemektedir. Bunun yerine aynı dizeye ait bireysel sembolleri birbirleriyle ilişkilendirerek kelimeler ve sayılar oluşturulmaktadır. Bu simgelerin dizeler halinde bir araya getirilmesi işlemi genel olarak gruplama olarak adlandırılır. Sembollerin

dizelere gruplanması, belgedeki sembollerin bulunduğu yere bağlıdır. Yeterli derecede yakın olduğu belirlenen semboller birlikte gruplandırılır.

Tüm bu işlemlerin sonunda, OCR belirli bir görüntüdeki metinsel karakterlerin, bir bilgisayarın işleyebileceği bir biçimde algılanması ve tanımlanması işlemidir [25].

1.7.3. JPEG-LS Algoritması

JPEG, günümüzde dijital görüntülerin sıkıştırılması için dünya çapında kullanılan bir standarttır. Standart, onu oluşturan komitenin adını almıştır ve gelişmeye yön vermeye devam etmektedir. Bu grup, JPEG (Joint Photographic Experts Group - Ortak Fotoğraf Uzmanları Grubu), ulusal standartlar organları tarafından belirlenen uzmanlardan ve görüntü ile ilgili çalışmaları yapan önde gelen şirketlerden oluşur. Komite, hareketsiz görüntü sıkıştırma için etkin bir standart seti üretmek için sıkıştırmaya umut verici yaklaşımları takip etme çabalarını bir araya getirme sorumluluğu ile yükümlüdür.

JPEG standardizasyon faaliyeti 1986 yılında başlatıldı ve Mart 1987'de komite tarafından değerlendirilmek üzere 12 öneri çıktı. İlk deneme, sıkıştırmanın ayrık kosinüs transformasyonuna (DCT) dayandırılacağı üzerinde fikir birliği sağladı. JPEG standardı, JPEG standardının hem kayıplı hem de kayıpsız sıkıştırma algoritmalarını tanımlar.

JPEG-LS kayıpsız sıkıştırma algoritması tahmine dayalı bir tekniktir. Bununla birlikte, aşağıdaki gibi önemli farklılıklar bulunmaktadır [17].

- JPEG-LS, geçerli pikselden geçen kenarların varlığını algılamaya çalışan ve buna bağlı olarak tahmini ayarlayan doğrusal olmayan bir tahmin mekanizması kullanır. Bu, tahmin aşamasında performansta belirgin bir iyileşme ile sonuçlanır.

- JPEG kayıpsız aritmetik gibi, JPEG-LS de kodlamadan önce tahmin hatalarının bazı basit fakat çok etkili bağlam modellemesini kullanır.

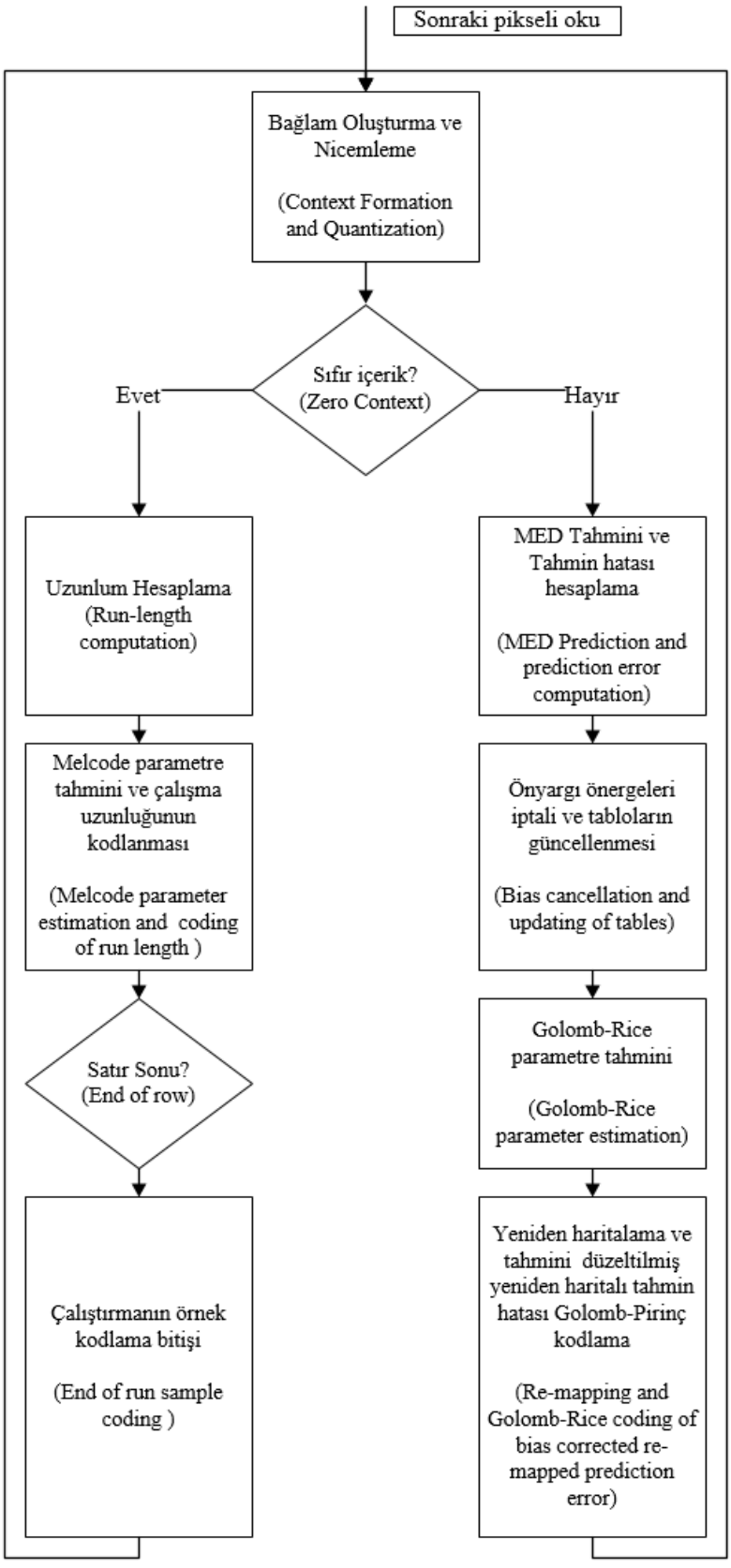
- Temel JPEG-LS, tahmin hatalarını kodlamak için Golomb-Rice kodlarını kullanır. Golomb-Rice kodları, tahmin hatalarının dağılımını karakterize eden belirli geometrik dağılımlar için Huffman kodlarını kullanırlar. Golomb-Rice kodları uzun süredir bilinmesine rağmen, JPEG-LS belirli bir bağlamda kullanılacak Golomb-Rice kodunun parametresini değişken olarak tahmin etmek için bazı yeni ve etkili teknikler kullanmaktadır.

- JPEG-LS, düşük entropi görüntülerini veya bölgelerini etkili bir şekilde kodlamak için, tek biçimli bir bölge ile karşılaşıldığında bir çalışma uzunluğu moduna geçerek basit bir alfabe genişletme mekanizması kullanır. Kullanılan RLC yine Golomb kodlarının bir uzantısıdır ve sıkıştırılabilir görüntüler için performansta oldukça belirgin bir iyileşme sağlar.

- Yüksek sıkıştırma oranları gerektiren uygulamalar için JPEG-LS, yeniden yapılandırılmış her pikseli orijinal değerinden yaklaşık olarak bir mesafede olmasını sağlayan, kayıplı olmayan bir mod sağlar.

JPEG-LS temel algoritmasına genel bir bakış Şekil 1.16. Temel JPEG-LS ‘ye genel bakışta gösterilmektedir.

Kayıpsız görüntü sıkıştırma algoritması olan JPEG-LS, pek çok açıdan birçok kayıpsız görüntü sıkıştırma algoritmasından daha iyi performans gösterir. Mükemmel hesaplamalı etkinlik ve kodlama özelliklerine sahiptir. Bu çalışmada, JPEG-LS algoritması aşağıdaki nedenlerden dolayı seçilmiştir. [26, 27]'den çıkarılan sonuçlara göre, JPEG-LS: i) önemli ölçüde daha düşük hesaplama karmaşıklığına sahiptir, ii) ortalama en iyi sıkıştırma oranına sahiptir, iii) sıkıştırma zamanı avantajına sahiptir.



Şekil 1.16. Temel JPEG-LS ‘ye genel bakış

1.7.4. Huffman Kodlama Algoritması

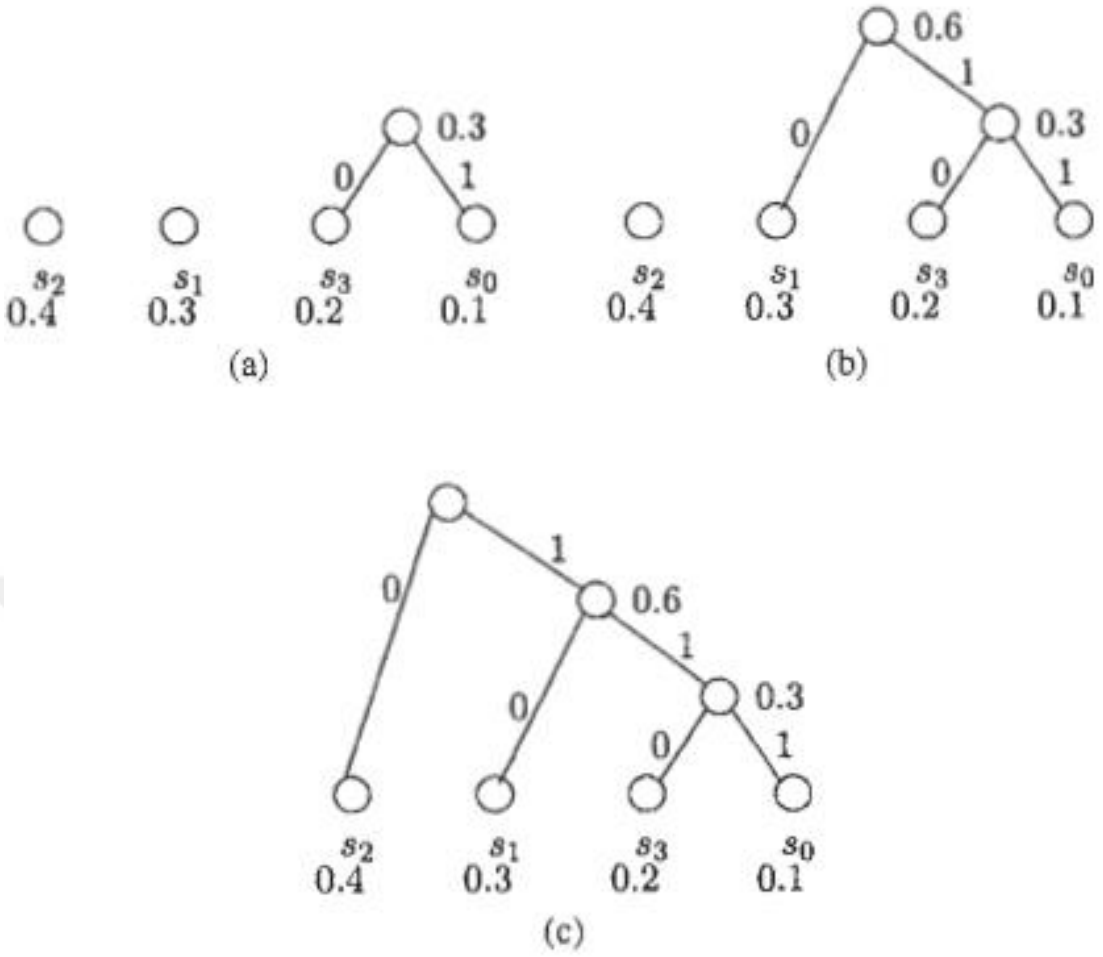
Bilgi teorisi ve bilgisayar bilimlerinde bir entropi kodlama algoritması olan Huffman kodlaması, kayıpsız veri sıkıştırma algoritmalarında yaygın olarak kullanılmaktadır. Huffman kodlaması istatistiksel verilere dayanır, bu da veri kalemlerinin tekrarlanma sıklığı anlamına gelir. Kodlama mekanizması esas olarak veri kümesindeki alfabe sıklığına bağlıdır [28]. Huffman kodlama ilkesi, daha az sayıda bitle veri kümesinde daha sık bulunan karakterleri temsil etmektir. Üretilen tüm kodlar bir kod defterinde saklanır. Başarılı kod çözme işlemi için, kod çizelgesi ve kodlanmış veriler tamamen saklanmalıdır [28].

Örneğin, Çizelge 1.4. Huffman kodlama veri örneğinde yer alan veriler kaynak alfabesi olarak düşünelim.

Çizelge 1.4. Huffman kodlama veri örneği

Kaynak sembol	Olasılık	Atanan kod kelimesi
S0	0.1	111
S1	0.3	10
S2	0.4	0
S3	0.2	110

Bu örnekteki alfabe kullanıldığında Huffman kodlama algoritması her karakteri ortalama 1,9 bitle sembolize etmektedir. Bu alfabeye ait çözüm ve yaklaşım Şekil 1.17. Kaynak alfabe için Huffman kod yapımı örneğinde gösterilmektedir.



Şekil 1.17. Kaynak alfabe için Huffman kod yapımı örneği

Şekil 1.17. Kaynak alfabe için Huffman kod yapımı örneğinde a) ilk tekrar, (b) ikinci tekrar ve (c) üçüncü tekrar adımlarını göstermektedir.

Huffman algoritması geliştirilmiş ve değişken huffman kodlaması geliştirilmiştir. Ancak bu çalışmada klasik huffman kodlama algoritması kullanıldığından değişken huffman kodlama algoritması kapsam dışı bırakılmıştır. Değişken huffman kodlaması algoritmasına ait detaylar [29-32]'da bulunabilir.

1.8. Bulut Tabanlı Sıkıştırma Probleminde Kullanılan Yöntemler

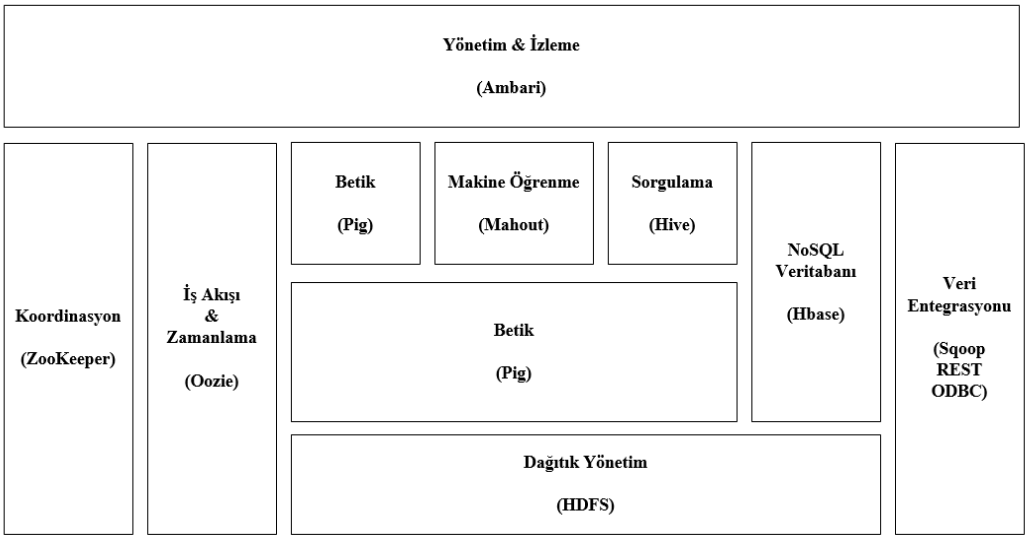
Çalışmanın bu bölümünde, görüntü bölütleme, optik karakter tanıma, JPEG-LS ve Huffman kodlama algoritması ile tek düğüm üzerinde gerçekleştirilen yaklaşım, bulut tabanlı bir sisteme uygulanmıştır. Çalışmanın bu kısmında görüntü işleme paketi, MapReduce/Hadoop, MongoDB ve GridFS yöntemlerinden bahsedilecektir.

1.8.1. Görüntü İşleme Paketi

Geliştirilen bulut tabanlı sıkıştırma yöntemi çalışmanın ilk bölümünde geliştirilen yöntemi ön işleme süreci olarak kullanmaktadır. Sisteme yüklenilen tüm medikal görüntüler öncelikle görüntü işleme paketi adı verilen ve ilk yöntemde önerilen sistemden geçirilerek işlenir. Ancak bu aşamada yapılan tüm işlemler MapReduce/Hadoop platformu üzerinde çalışmaktadır.

1.8.2. MapReduce/Hadoop

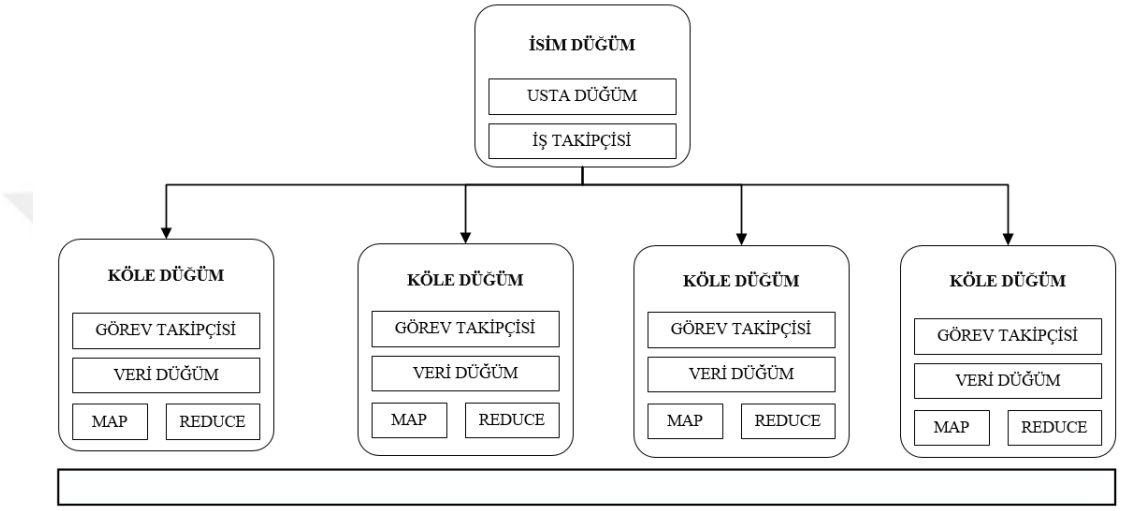
MapReduce/Hadoop, bilgisayar bilimleri alanında devrim yaratan ve büyük verilerin en yeni konularından biri olan veri işleme ve analiz teknolojisidir. Gelişen platformlar, Şekil 1.18. Apache Hadoop ekosisteminde gösterilen Apache Hadoop ekosistemi kurulmasını sağlamıştır.



Şekil 1.18. Apache Hadoop ekosistemi

Hadoop, yaygın olarak kullanılan metin arama kitaplığı olup Apache Lucene geliştiricisi Doug Cutting tarafından geliştirilmiştir. Hadoop'un kökeni, Lucene projesinin bir parçası olan açık kaynaklı bir web arama motoru olan Apache Nutch'tir. Nutch, 2002'de faaliyete başlamıştır ve çalışan bir tarayıcı ve arama sistemi ile piyasaya çıkmıştır. Bununla birlikte, geliştiriciler mimarilerinin internetteki milyarlarca sayfaya ölçeklenemeyeceğini fark etmişlerdir. Çözüm ise 2003'te Google'ın dağıtılmış dosya sisteminin, Google'daki üretimde kullanılan GFS adlı mimarisini tanımlayan bir makalenin yayınlanmasıyla bulunmuştur [33]. Geliştirilen bu dosya sistemi web tarama ve indeksleme işleminin bir parçası olarak üretilen çok büyük dosyalar için depolama ihtiyaçlarını çözecekti. GFS sayesinde sistemi yönetim için harcanacak zamana gerek kalmayacaktır. Nutch'in geliştiricileri, 2004 yılında, açık kaynaklı bir uygulama olan Nutch dağıtık dosya sistemi yazmaya başlamıştır. 2004'te Google, MapReduce'u tanıtan makaleyi dünyaya yayınladı [34]. Bunun üzerine Nutch geliştiricilerinin Nutch'de bir MapReduce uygulaması vardı ve 2005'in başında tüm önemli Nutch algoritmaları MapReduce ve NDFS'yi kullanacak şekilde taşınmıştır. Nutch'teki NDFS ve MapReduce uygulaması arama alanının ötesinde de uygulanabilir olduğu görülmüştü ve Şubat 2006'da Lucene'nin Hadoop adlı bağımsız

bir alt projesi oluşturulmuştur. Ardından Doug Cutting Yahoo'da çalışmaya başladı ve Şubat 2008 yılında tüm arama dizinleri 10000 çekirdekli bir Hadoop kümesi tarafından yönetilmeye başlandığı açıklanmıştır [35]. Hadoop Şekil 1.19. Hadoop usta/köle mimarisinde gösterildiği gibi kendi içerisinde usta/köle (master/slave) ilişkisi içerisinde çalışan bir yapıya sahiptir.



Şekil 1.19. Hadoop usta/köle mimarisi

MapReduce, büyük miktarda veriyi işleme için Google tarafından geliştirilen bir programlama modelidir ve birincil hedefi milyarlarca web sayfasının indekslenmesidir [36, 37]. Hadoop'un sağladığı paralel işleme avantajından yararlanmak için, sorguyu bir MapReduce işi olarak ifade etmemiz gerekmektedir [37].

MapReduce, Java ile geliştirilmiş iki seviyeli bir paralel veri işleme için bir programlama modelidir: Harita (Map) ve Küçültme (Reduce). Verilerin kendisi küçük parçalara bölünür ve MapReduce, hesaplamaları bulunduğu yere (Map) dağıtmak ve sonuçları özetlemek için kullanılır (Reduce). Hadoop, MapReduce ve Hadoop Dağıtılmış Dosya Sistemi (Hadoop Distributed File System - HDFS) 'den oluşan açık kaynak kodlu bir çözümdür [38, 39]. Hadoop, bir ad düğümünden ve birçok veri

düğümünden oluşan dağıtılmış bir mimari platformdur. Son yıllarda, bu teknoloji sıklıkla sağlık hizmetleri alanında aşağıdaki nedenlerle sıkça kullanılmıştır. i) yazılım çerçevesi geliştirilmesi, ii) tıbbi büyük veri işleme sistemlerinin geliştirilmesi ve iii) büyük ölçekli tıbbi görüntülerin analizi [38, 39].

1.8.3. MongoDB ve GridFS

İlişkisel veri tabanı yönetim sistemleri (Relational Database Management Systems - RDBMS) günümüzde yaygın olarak kullanılan veri tabanı yönetim sistemleri (Database Management Systems - DBMS) haline gelmiştir. Bununla birlikte, bu yöntemler büyük veri işleme gibi uygulamalarda etkili değildir. Bu nedenle, NoSQL DBMS geliştirilmiştir. İlişkisel DBMS'ler veri tabanı pazarında halen hakim olmakla beraber NoSQL DBMS platformları ve kullanımı da günden güne artmaktadır [40]. En yaygın kullanılan belge tabanlı NoSQL veri tabanı olan MongoDB, bir açık kaynak DBMS'dir. MongoDB, web uygulamaları ve internet altyapısı için tasarlanmış bir veri tabanı yönetim sistemidir. Veri modeli ve kalıcılık stratejileri yüksek okuma ve yazma çıktıları ve otomatik yerine çalışma ile kolaylıkla ölçeklenebilme yeteneği için oluşturulmuştur. Bir uygulamanın sadece bir veritabanı düğümü veya düzinelerce veri tabanı düğümü gerektirmesi durumunda, MongoDB şaşırtıcı derecede iyi bir performans sağlamaktadır. Belge tabanlı bir veri modelinin zengin, hiyerarşik veri yapılarını temsil edebileceği göz önüne alındığında, ilişkisel veri tabanları tarafından dayatılan karmaşık çoklu tablo birleştirmeleri olmadan çalışmak avantaj sağlamaktadır [41]. Sonuç olarak MongoDB güçlü, esnek ve ölçeklenebilir genel amaçlı bir veri tabanıdır. İkincil dizinler, aralık sorguları, sıralama, toplamalar ve coğrafi dizinler gibi özelliklerle ölçeklendirme özelliğini birleştirir [42].

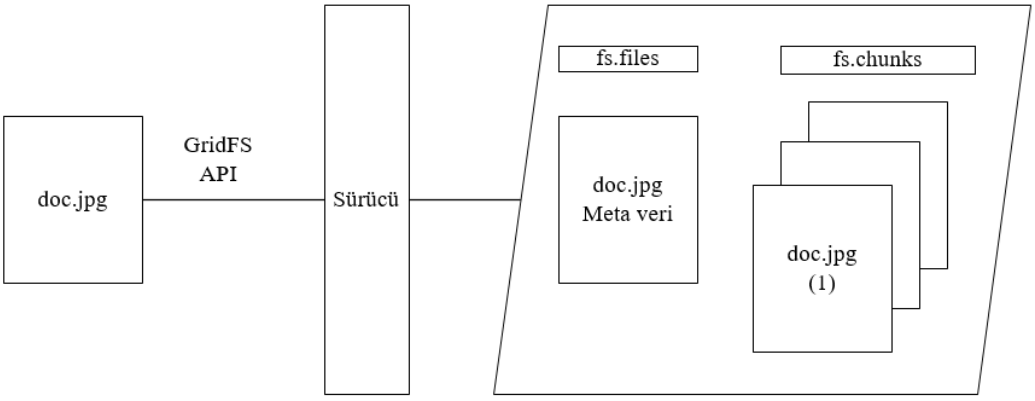
DbEngines'in araştırmasına göre MongoDB tüm DBMS'ler arasında beşinci sırada ve NoSQL DBMS'ler arasında birinci sırada yer almaktadır [43]. MongoDB şema tabanlı değildir ve her belge BSON formatında saklanır. Bu biçimde, bir nesne sırasız isim / değer çifti grubudur. MongoDB‘de tanımlanmış bir tablo yapısı olmadığından BSON formatlı herhangi bir belge eklenebilir. Ayrıca, MongoDB dağıtılmış bir ortamı destekler ve çeşitli şekillerde konuşlandırılabilir. Üç farklı dağıtım türü vardır:

çoğaltma kümesi, bağımsız ve keskin küme [44]. Bu çalışmada, veri dağıtımında parçalanmış küme dağıtım tipi kullanılır. Parçalanmış küme dağıtımında, veriler parçalanır ve birden çok sunucuda depolanır.

GridFS, değişken boyuttaki dosyaları MongoDB'de depolamak için kullanılan bir kuraldır. GridFS özelliği tüm resmi sürücüler ve MongoDB'nin mongofiles aracı tarafından uygulanır, böylece platformlar arasında tutarlı bir erişim sağlanır. GridFS, büyük ikili nesneleri veri tabanında depolamak için kullanışlıdır. Genellikle bu nesneye hizmet etmek için yeterince hızlıdır ve depolama yöntemi akış için elverişlidir [41].

GridFS, dosyaları depolamak ve bir MongoDB veri tabanından kolayca almak için kullanılan yerleşik bir MongoDB fonksiyonudur. Bir veri formatı değil, büyük boyutlara sahip dosyaları MongoDB veri tabanında saklamak avantajlı bir özelliktir. GridFS, büyük bir dosyayı küçük, 256 KB parçalara bölerek ve her parçayı ayrı bir belge olarak depolayarak çalışmaktadır. Varsayılan olarak, bu bölümler fs.chunks adlı bir koleksiyonda saklanmaktadır. Kısımlar yazıldığında, dosyanın meta verileri, fs.files adlı başka bir koleksiyondaki tek bir belgede saklanmaktadır [41].

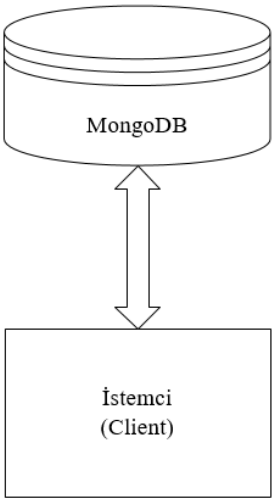
GridFS mantığını daha iyi anlamak için örnek bir resim dosyası üzerinde mimarisi Şekil 1.20. GridFS mimarisinde gösterilmiştir.



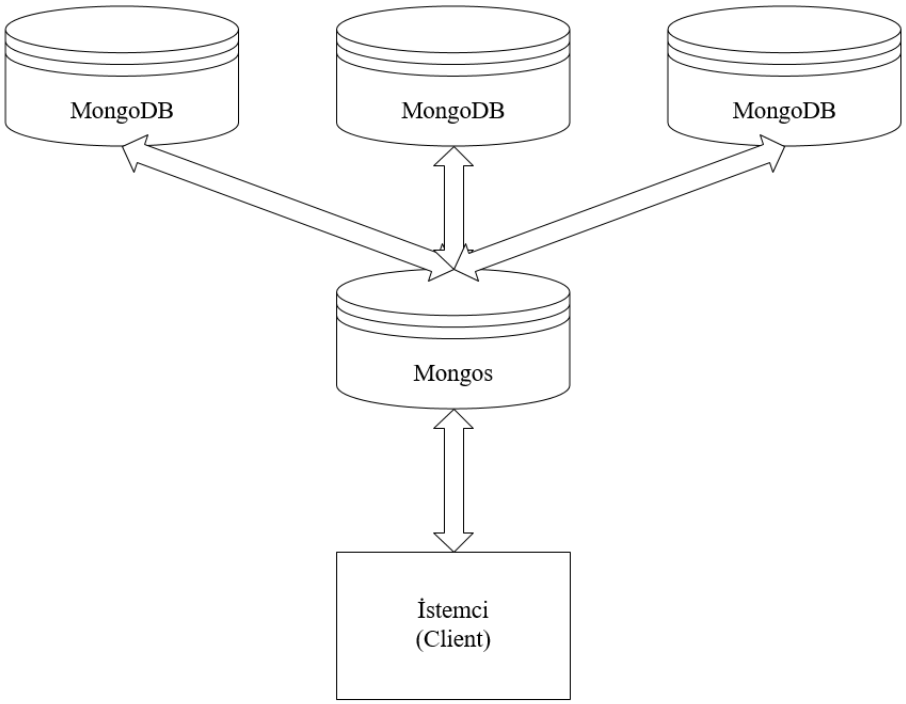
Şekil 1.20. GridFS mimarisi

MongoDB dağıtık mimariyi desteklemektedir. Sharding, verilerin makineler arasında bölünme sürecine işaret eder; Bölümleme terimi bazen bu kavramı tanımlamak için de kullanılır. Her makineye bir veri alt kümesi koyarak, daha az güçlü daha az makine ile daha fazla veri depolamak ve daha büyük veya daha güçlü makinelere ihtiyaç duymadan daha fazla yük idare etmek mümkün hale gelir. MongoDB, mimariyi uygulamadan uzaklaştıracak ve böyle bir sistemin yönetimini basitleştirecek otomatik Sharding mimarisini desteklemektedir. MongoDB, uygulamanın bir ölçüde bağımsız bir MongoDB sunucusuyla konuşmadığını göz ardı etmesini sağlar. Operasyon tarafında, MongoDB, verileri parçalardan dengelemeyi otomatikleştirir ve kapasiteyi artırmayı ve azaltmayı kolaylaştırır [42].

MongoDB'de Sharding mimarisinin tercih edilmesi, birçok makinenin (parçalı) bir küme oluşturmasına ve koleksiyonun parçalamasına ve her parçada bir veri alt kümesi oluşturulmasına olanak tanır. Bu, uygulamanın bağımsız bir sunucu veya çoğaltma kümesinin kaynak sınırlarının ötesinde büyümesine izin verir. Parçalamanın hedeflerinden biri, 5, 10 veya 1.000 makinenin bir kümesinin uygulama için tek bir makine gibi görünmesini sağlamaktır. Sharding mimarisi kullanan ve kullanmayan MongoDB mimarileri Şekil 1.21. Sharded olmayan bağlantı modeli ve Şekil 1.22. Sharded Bağlantı Modelinde gösterilmektedir [41, 42, 44].



Şekil 1.21. Sharded olmayan bağlantı modeli



Şekil 1.22. Sharded Bağlantı Modeli

MongoDB literatürde sağlık alanında kullanılmakta ve tatmin edici sonuçlar elde edilmektedir [44].

Bu çalışmada performansı ve getirdiği avantajlar nedeniyle MongoDB tercih edilmiştir, ayrıca büyük boyutlu dosyalara uygun bir sistem olabilmesi için GridFS yerleşik fonksiyonu da kullanılmıştır.

1.9. Çalışmanın Amacı ve Katkıları

Bu çalışmanın amacı her geçen gün artan boyuta sahip olan medikal görüntülerin sıkıştırılması ile gerek depolama gerek iletim maliyetlerinin azaltılmasıdır. Yapılan bu çalışmada literatürde sıkça kullanılan yöntem ve algoritmalar kullanılmıştır. Bu çalışma, sıkıştırma yöntemi geliştirilmesi ve geliştirilen yöntemin bulut tabanlı sisteme uygulanması olmak üzere iki aşamalıdır.

İlk olarak ROI tabanlı sıkıştırma yöntemi geliştirilmiştir. Bu katmanda ham halde alınan tüm medikal görüntüler tek düğüm üzerinde çalışan belirli yöntem ve algoritmalar ile sıkıştırılmıştır. Bu çalışma sonucunda aşağıdaki sonuçlar ve katkılar elde edilmiştir.

- Yeni bir sıkıştırma yaklaşımı geliştirilmiştir.
- Medikal görüntülerin ROI bölgesi ortalama %30 oranında sıkıştırılmıştır.
- Medikal görüntülerin Non-ROI bölgesi %92,12 ile %97,84 arasında sıkıştırılmıştır.
- Geliştirilen bu yöntem ile literatürde bulunan son teknolojiden ortalama %83,80 oranında daha başarılı sıkıştırma sağlamıştır.

Yapılan diğer çalışma ile ilk aşamada tek düğüm üzerinde uygulanan yöntem bulut tabanlı olarak geliştirmiş ve ilk çalışmaya göre daha başarılı performans elde edilmiştir. Yapılan çalışma sonucunda elde edilen sonuçlar ve katkılar aşağıdaki gibidir.

- Medikal görüntülerin işlenmesi aşamasındaki süreçler MapReduce programlama modeli ile Hadoop platformu üzerinde yapılmıştır. Tek düğüm üzerinde 15,599 saniyede tamamlanan işlemler önerilen bulut tabanlı sistem sayesinde 8,153 saniyede gerçekleştirilmiştir.
- Geliştirilen sistem otomatik replikasyon ve bölümleme mekanizmasına sahip olduğundan herhangi bir kontrole gerek kalmaksızın hataya ve veri kaybına dayanıklı bir sistemdir.
- Sıkıştırma sonucu elde edilen veriler bir NoSQL veri tabanı olan dağıtık MongoDB veri tabanı üzerinde saklanmıştır.
- Efektif bir arama yaklaşımı geliştirilmiştir.

2. LİTERATÜR İNCELEMESİ

Jacob tarafından 1997 yılında yapılan çalışmada medikal görüntü sıkıştırma yöntemi geliştirilmiştir [45]. Bu çalışmada medikal görüntülerin belirli bölgelerinin kalan bölgelerden daha önemli olduğu belirtilmiş ve ROI tabanlı bir sıkıştırma yöntemi geliştirilmiştir. Çalışmada ROI bölgesine S-dönüşümü ile kayıpsız kodlama uygulanmış, non-ROI bölgesine ise kayıplı yöntem olan wavelet kodlama uygulanmıştır.

Zhiyong tarafından 2015 yılında ROI tabanlı bir medikal görüntü sıkıştırma yöntemi geliştirilmiştir [23]. Çalışmada medikal görüntüler aktif çevre modeli kullanılarak ROI ve non-ROI bölgelerine ayrılmıştır. Veri kaybına toleransı olmayan ROI bölgesine JPEG-LS kayıpsız sıkıştırma algoritması uygulanmıştır. Non-ROI bölgesine ise kayıplı wavelet sıkıştırma algoritması uygulanmıştır. Bu sayede ROI bölgesi kayıpsız geri dönüştürülebilmiş ve non-ROI bölgesi ise yüksek sıkıştırma oranları ile sıkıştırılmıştır. Çalışma sonucunda ROI bölgesi 2,2556 ile 3,9161 arasında sıkıştırma oranına sahip olmuştur. Non-ROI bölgesinde ise 10,1290 ile 13,0457 arasında sıkıştırma sonuçları elde edilmiştir.

Göktürk tarafından 2001 yılında yapılan çalışmada ise kolon tomografisi görüntülerine ROI tabanlı bir sıkıştırma yöntemi geliştirilmiştir [46]. Çalışmada ROI ve non-ROI bölgesi 3-D morfolojik görüntü işleme teknikleri ile ayrılmıştır. Daha sonra, hareket vektörleri görüntünün her bloğu için kodlanmıştır. Son olarak, gerçek görüntü ve hareket öngörülen görüntü arasındaki hata, ROI blokları için kodlanmıştır.

Kaur tarafından 2015 yılında yapılan çalışmada ise tele tıp uygulamaları için ROI tabanlı medikal görüntü sıkıştırma yöntemi önerilmiştir [47]. Tele tıp ve medikal görüntüleme teknolojileri geliştiğinden depolama ve iletimde bant genişliği gereksinimi artmıştır. Çalışmada görüntü bölütleme uygulanarak medikal görüntü ROI ve non-ROI bölgelerine ayrılmıştır. ROI bölgesine bağlam ağacı ağırlıklı kayıpsız (Context tree weighting lossless) sıkıştırma yöntemi, Non-ROI bölgesine ise fraktal kayıplı sıkıştırma (Fractal lossy compression) yöntemi uygulanmıştır.

Iqbal tarafından 2015 yılında kanser hastalarına özel bir model geliştirilmiştir [48]. Geliştirilen bu sistem üzerinden medikal görüntüler anime edilmiştir. 782 milyon hasta verileri kategorize edilmiş ve işlenmiştir. Medikal görüntüler büyük veriler olarak işlenmiş ve öngörü aşamalarında kullanılmıştır.

Eman tarafından 2012 yılında yapılan bulut tabanlı e-sağlık literatür taramasında bulut bilişime ait tüm fırsatlar ve zorluklar sunulmuştur [49]. Yapılan çalışma sonucunda bulut bilişimin sağlık alanına uygulanması öncesi, aşaması ve sonrasında yaşanabilecek fırsatlar ve zorluklar teknik ve teknik dışı kriterlerle incelenmiştir. Sonuç olarak maliyetlerin düşürülmesi, daha iyi entegrasyon ve tıbbi kayıt alışverişi sağlamak, süreçleri hızlandırmak, teşhisi geliştirmek, tıbbi araştırma faaliyetlerini desteklemek ve idari işlemleri basitleştirmek için birden çok tarafın bilgi havuzundan yararlanmasına izin verilmesi, sağlık bilgi sistemlerinin kullanılabilirliğini, ölçeklenebilirliğini ve esnekliğini arttırmak gibi avantajlar sunulmuştur.

Kagadis tarafından 2013 yılında yapılan çalışmada ise medikal görüntülerin bulut bilişime uygulanması önerilmiştir [50]. Çalışmada bulut servisleri ve tipleri tanımlanmıştır. Ayrıca bulut bilişimde veri güvenliği, veri yönetimi ve veri transferi tartışılmıştır. Tüm bunlara ek olarak bulut maliyetleri de çalışmaya eklenmiştir. Bulut bilişime uygun PACS sistemleri ve avantajları çalışmada bulunmaktadır. Sonuç olarak bulut bilişimin hastane sistemlerinde kullanılmasının büyük avantajları olduğu bildirilmiştir. Ancak maliyet ve büyüklük düşünüldüğünde yapılacak geçişlere dair uyarılar ve öneriler de yapılmıştır.

Chang tarafından 2016 yılında yapılan çalışmada büyük verilerin bulut ve bulut olmayan yöntemlerle depolanmasını karşılaştıran bir model geliştirilmiştir [51]. Çalışmada bulut ve bulut olmayan depolama alanlarına ait karşılaştırmanın adil olduğundan emin olmak zor olabilir fikrinden yola çıkılmıştır. Çalışmada böyle bir karşılaştırmanın kurulma süreci ve kullanılan metrik incelenmiştir. Çalışmada bulut ve bulut olmayan sistemler üzerinde performans karşılaştırmaları, verimlilik ve performansın iyileştirilmeleri değerlendirilmiştir. Deneylerden önce, ağ gecikmesi, dosya boyutu ve iş başarısızlıkları, performansı düşüren etkenler olarak tanımlanmış ve etkilerini anlamak için deneyler yapılmıştır. Çalışmada örgütsel sürdürülebilirlik

modellemesi (Organizational Sustainability Modeling – OSM), deneylerin öncesinde, sırasında ve sonrasında, adil karşılaştırmaların yapılmasını sağlamak için kullanılmıştır. 40 farklı test yapılmıştır. İlk test 1 GB boyutunda 10000 dosyanın transfer edilmesi ve yedeklenmesi, ikinci test ise 10 GB boyutunda 1000 dosyanın transfer edilmesi ve yedeklenmesi olarak düzenlenmiştir. Sonuçlar, bulut üzerindeki gerçek ve beklenen yürütme süresinin bulut olmayan sisteme göre daha düşük olduğunu göstermiştir. İkincisi, bulut üzerinde gerçek ve beklenen yürütme süresi arasında %99'dan fazla tutarlılık bulunmuştur, ancak bulut olmayan sistemde karşılaştırılabilir bir tutarlılık bulunamamıştır. Son olarak, bulut üzerindeki verimlilik iyileşmesinin bulut olmayan durumdan daha yüksek olduğu ifade edilmiştir. Sonuç olarak bulut sistem önerisinde bulunulmuştur.

Lin tarafından 2015 yılında yapılan çalışmada büyük tıbbi veriler üzerinden evde tanı servisi için bulut tabanlı bir yazılım çerçevesi geliştirilmiştir [52]. Yapılan çalışmada, Lucene tabanlı bir dağıtılmış arama kümesi, eşzamanlı ve ölçeklenebilir tıbbi kayıt alımı, veri analizi ve gizlilik korumasını desteklemek üzere tasarlanmıştır. Dahası, tıbbi kayıt geri alımını hızlandırmak, çevrimdışı veri depolama ve dizin oluşturma için bir Hadoop kümesi uygulanmıştır. Yapılan çalışmayla hastaların hangi hastalığa yakalandıklarını bulmalarına yardımcı olmak için benzer geçmiş tıbbi kayıtların yanı sıra bir hastalık belirti dizisi elde edilmiştir. Ayrıca çalışmada, bir prototip sistemi tasarlanmış ve önerilen sistemin ölçeklenebilirliğini ve verimliliğini göstermek için çalışan bir örnek sunulmuştur.

Istephan tarafından 2016 yılında yapılan çalışmada büyük veri kullanarak yapılandırılmamış tıbbi görüntü sorgulama incelemesi yapılmıştır [53]. Çalışmanın amacı, yapılandırılmamış verilerin sınırsız yollarla verimli bir şekilde sorgulanmasına yönelik böyle bir çerçeveye sahip olmanın fizibilitesini uygulamak ve incelemektir. Fizibilite çalışması özellikle epilepsi alanında yapılmıştır. Önerilen model, bir sorguyu iki aşamalı olarak değerlendirmektedir. Faz 1'de, klinik veri ambarını filtrelemek için yapısal veriler kullanılmıştır. Faz 2'de, özellik çıkarma modülleri, sorguyu tamamlamak için yapılandırılmamış veriler üzerinde Hadoop vasıtasıyla dağıtık bir mimari üzerinde yürütülmüştür. Önerilen çerçeve, yapılandırılmamış tıbbi verilerin

içeriğini tıbbi uzmanların kullanabileceği etkili ve sınırsız bir yolla açarak veri odaklı yaklaşımıyla literatüre katkı sağlamaktadır.

Yao tarafından 2015 yılında yapılan çalışmada Hadoop tabanlı tıbbi büyük veri işleme sistemi tasarlanmış ve geliştirilmiştir [54]. Apache Hadoop MapReduce, geniş kümeleriyle büyük miktarda veriyi güvenilir, hataya dayanıklı bir şekilde paralel olarak işleyen uygulamaları geliştirmek için kullanılan bir yazılım çerçevesidir. Yapılan çalışmada, Hadoop tabanlı tıbbi büyük veri işleme sisteminin pratik bir örneği incelenmiştir. Bu sistem tıbbi büyük verileri akıllıca işlemek ve hastane bilgi sistemi kullanıcı davranışlarının bazı özelliklerini ortaya çıkarmak için geliştirilmiştir. Çalışmada farklı hastane bilgi sistemleri tarafından günlük iş için üretilen çeşitli verilere ilişkin kullanıcı davranışları incelenmektedir. Dağıtık MapReduce algoritmalarını çalıştırmak için beş düğümlü bir Hadoop kümesi de kurulmuştur. Geliştirilen sistem dağıtık algoritmalarla, sağlık hizmetleri ve klinik araştırmalarda büyük tıbbi verilere sahip tek bir düğümle karşılaştırıldığında etkili veri işlemeyi kolaylaştırmıştır.

Malekabadi tarafından bulut bilişiminde büyük sağlık verilerini depolamak ve almak için etkili bir model önerilmiştir [55]. Bu çalışma, sağlık verilerinin depolanması için NoSQL veri tabanlarına dayanan modelin sunulmasını amaçlamıştır. NoSQL veri tabanlarının farklı türlerine rağmen belge tabanlı DBMS'ler seçilmiştir. Sunulan model, dağıtım özelliklerine erişmek için bulut ortamında uygulanmıştır. Ardından veriler Shard özelliğini kullanarak veri tabanında dağıtılmıştır. Modelin etkinliği, önceki veri modeli olan ilişkisel veri tabanı ile karşılaştırıldığında, sorgu süresi, veri hazırlığı, esneklik ve genişletilebilirlik parametreleri göz önünde bulundurularak değerlendirilmiştir. Sunulan modelin performansı, esnekliği, veri hazırlığı ve genişletilebilirlik durumları SQL Server'dan daha iyi olarak tespit edilmiştir. Sonuç olarak, önerilen model, sağlık verilerini işleme için ilişkisel veri tabanlarından daha etkilidir.

Jongseong tarafından 2016 yılında yapılan çalışmada adli soruşturma belge deposu veri tabanı sistemi MongoDB örnek çalışması sunulmuştur [44]. Çalışmada NoSQL DBMS ile adli soruşturma belge deposu çerçevesi önerilmiştir. Bu vaka çalışması için,

deneysel bir ortamda bir suç senaryosu oluşturulmuş ve MongoDB için adli prosedür ve teknik yöntemler önerilmiştir. Çalışma sonuç olarak dağıtılmış bir ortamda kanıtları depolayan gerçek sunucuları ve işlem yeniden yapılandırma yöntemini belirlemek, günlük analizi kullanarak ve silinmiş verileri MongoDB veri dosyası yapısından kurtarmak da dahil olmak üzere, MongoDB için önemli teknik araştırma yöntemleri önermektedir.

Xin tarafından 2014 yılında yapılan çalışmada NoSQL sistemine dayalı CNEOST görüntü veritabanı tasarımı ve uygulanması geliştirilmiştir [56]. Çin yakınında Dünya nesnesi araştırması teleskobu, Çin'deki en büyük schmidt teleskop olup, 2006 yılında ilk ışığı gördüğünden beri 3 TB'tan fazla astronomik görüntü verisi elde etmiştir. CCD kameranın 2013 yılında yükseltilmesinden sonra her yıl 10 TB'ın üzerinde veri elde etmektedir. Gereksinim analizine dayanarak, bir görüntü yönetim sistemi, ilişkisel olmayan veritabanı kullanılarak tasarlanmış ve uygulanmıştır. Çalışma sonucunda yönetim verimliliği geliştirilmiş, bakım problemleri azalmış ve astronomik araştırmalar için daha iyi bir servis sunulmuştur.

Liao tarafından 2016 yılında yapılan çalışmada SQL ve NoSQL veri tabanları arasında sorgulama ve dönüşüm için veri bağdaştırıcısı geliştirilmiştir [57]. Bu çalışmada, RDMS ve NoSQL sistemlerinde çoklu yapılandırılmış verilerin otomatik olarak dönüştürülmesini mümkün kılmak için veri bağdaştırıcısı sunulmaktadır. Önerilen veri bağdaştırıcısı ile, hibrit veritabanı sistemlerini oluşturmak için sorunsuz bir mekanizma geliştirilmiştir. Önerilen veri bağdaştırıcısı ile, hibrit veritabanı sistemleri elastik bir şekilde gerçekleştirilebilir, diğer bir deyişle, verilerin boyutuna bağlı olarak erişim RDMS veya NoSQL olabilmektedir.

Li tarafından 2013 yılında yapılan bir çalışmada SQL ve NoSQL veri tabanlarının performans karşılaştırması yapılmıştır [58]. Çalışma sonuçlarına göre, tüm NoSQL veri tabanları SQL veri tabanlarından daha iyi performans göstermediğini bildirilmektedir. Bazı sonuçlar ise çok daha kötüdür. Ve her veritabanı için performans uygulanan operasyona göre değişmektedir. Veritabanı tarafından kullanılan veri modeli ile performans arasında küçükte olsa bir korelasyon vardır. Çalışma sonucunda veri miktarları arttıkça NoSQL veri tabanı hızının arttığı görülmektedir.

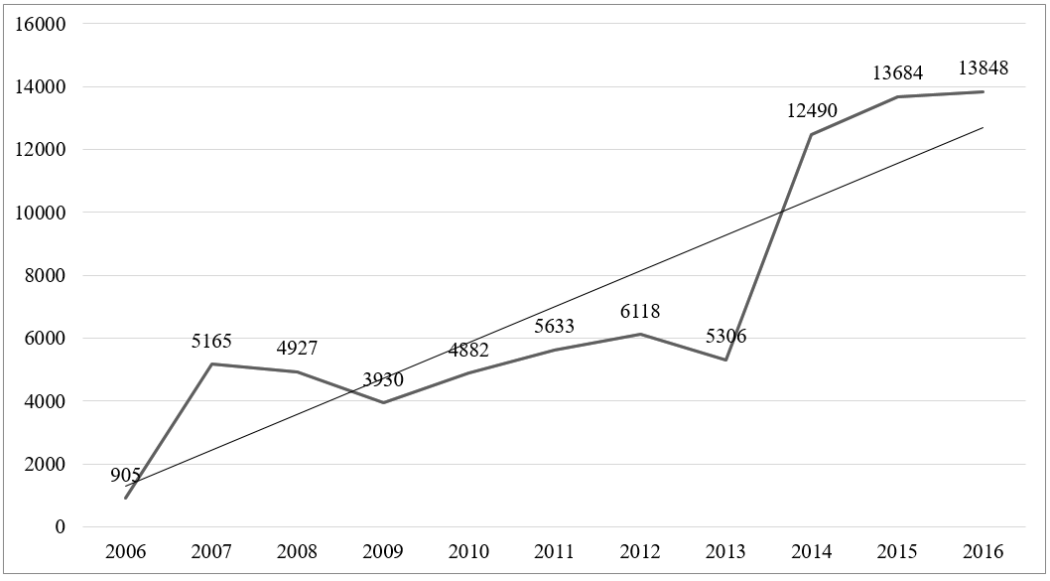
3. MATERYAL VE YÖNTEM

3.1. Veri Seti

Kırıkkale Üniversitesi Tıp Fakültesi verilerine göre;

- 2006 ile 2016 yılları arasında kurumun 76888 adet MR görüntüsü bulunmaktadır.
- PACS kurulumu 2013 yılında yapılmıştır.
- 2013 ile 2017 yılları arasında 51446 MR görüntüsü bulunmaktadır.

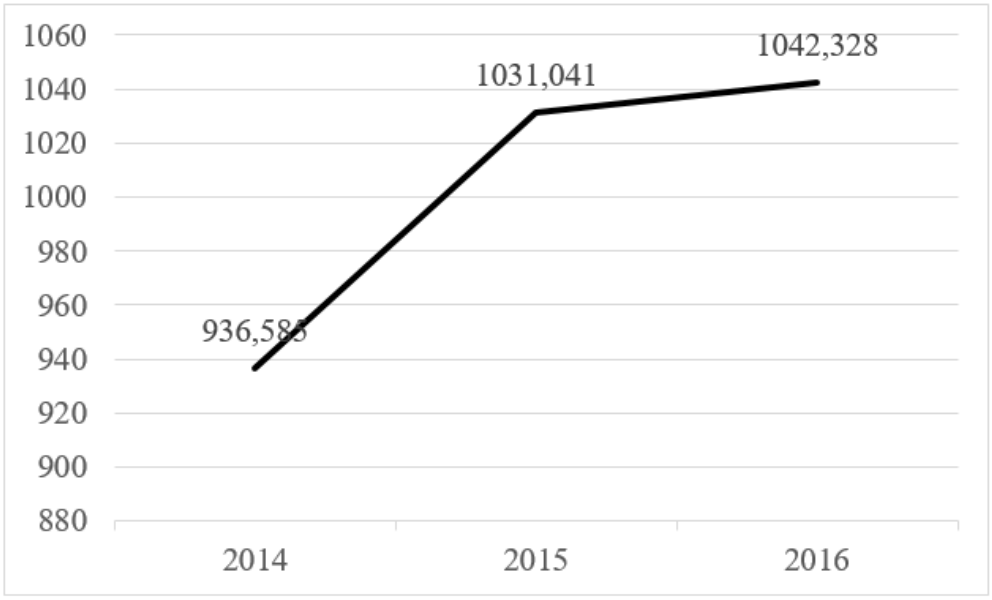
Yıl bazlı detaylı MR görüntü sayıları Şekil 3.1. MR görüntü sayılarında gösterilmektedir.



Şekil 3.1. MR görüntü sayıları

Görüldüğü üzere yıl bazında MR sayıları her geçen gün artmıştır, özellikle son 10 yılda yaklaşık 13 kat artış göstermiştir.

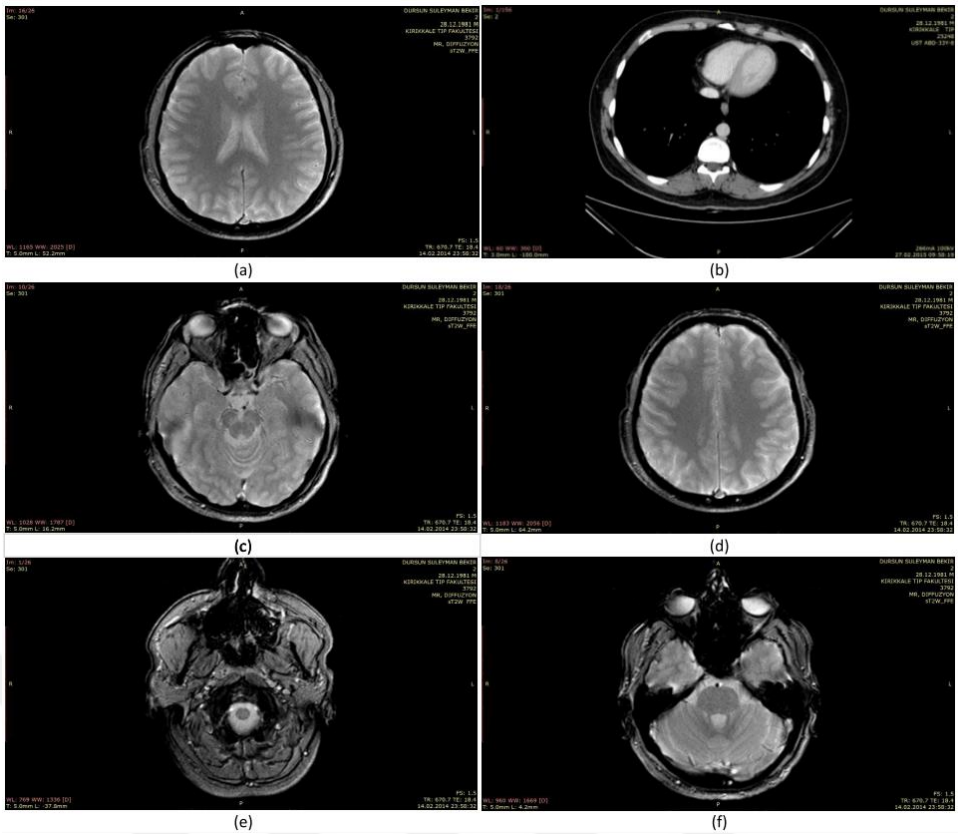
Çekilen bu MR görüntüleri saklanmakta ve arşivlenmektedir. MR görüntülerinin son 3 yılda kullandıkları depolama boyutu aşağıda gösterilmektedir.



Şekil 3.2. MR görüntü boyutu

Şekil 3.2. MR görüntü boyutu şeklinde GB cinsinden görülen görüntü boyutları yalnızca kullanılan ortam için olup ayrıca yedekleme için de aynı oranlarda depolama alanı ayrılmaktadır. Yani gösterilen tüm verilerin iki kat yer ihtiyacı bulunmaktadır.

Ancak geliştirilen sistemin test edilmesi ve değerlendirilmesi aşamalarında Şekil 3.3. Test görüntülerinde gösterilen ve gerçek bir hastaya ait olan 6 adet beyin MR görüntüsü kullanılmıştır.



Şekil 3.3. Test görüntüleri

Kullanılan beyin MR test görüntülerine ait detaylar Çizelge 3.1. Test görüntü detaylarında görüldüğü gibidir.

Çizelge 3.1. Test görüntü detayları

Görüntü	Görüntü bölütü	Oran (%)	Dosya boyutu (MB)	Toplam dosya boyutu (MB)
Test görüntüsü (a)	ROI	29,39	2,560	8,712
	Non-ROI	70,61	6,152	
Test görüntüsü (b)	ROI	43,23	3,991	9,231
	Non-ROI	56,77	5,240	
Test görüntüsü (c)	ROI	32,12	2,777	8,647
	Non-ROI	67,88	5,870	
Test görüntüsü (d)	ROI	28,76	2,318	8,059
	Non-ROI	71,24	5,741	
Test görüntüsü (e)	ROI	27,33	2,318	8,482
	Non-ROI	72,37	5,971	
Test görüntüsü (f)	ROI	36,21	3,390	9,361
	Non-ROI	63,79	5,971	

3.2. Geliştirme Ortamı

Çalışma sırasında iki bilgisayar kullanılmıştır. Sıkıştırma probleminin geliştirilmesi aşamasında Intel Core i7-4500U işlemci, 16GB RAM ve 1 TB SATA HDD'ye sahip bir dizüstü bilgisayar kullanılmıştır.

Bulut tabanlı sıkıştırma probleminin geliştirilmesi aşamasında Intel Xeon E5-2620 v4 2.10 GHz 2 adet işlemci (16 çekirdekli) sunucu, 64 GB RAM, 1 TB SATA HDD ve 512 GB SSD özelliklerine sahip bir iş istasyonu kullanılmıştır.

3.3. Sıkıştırma Problem Tanımı

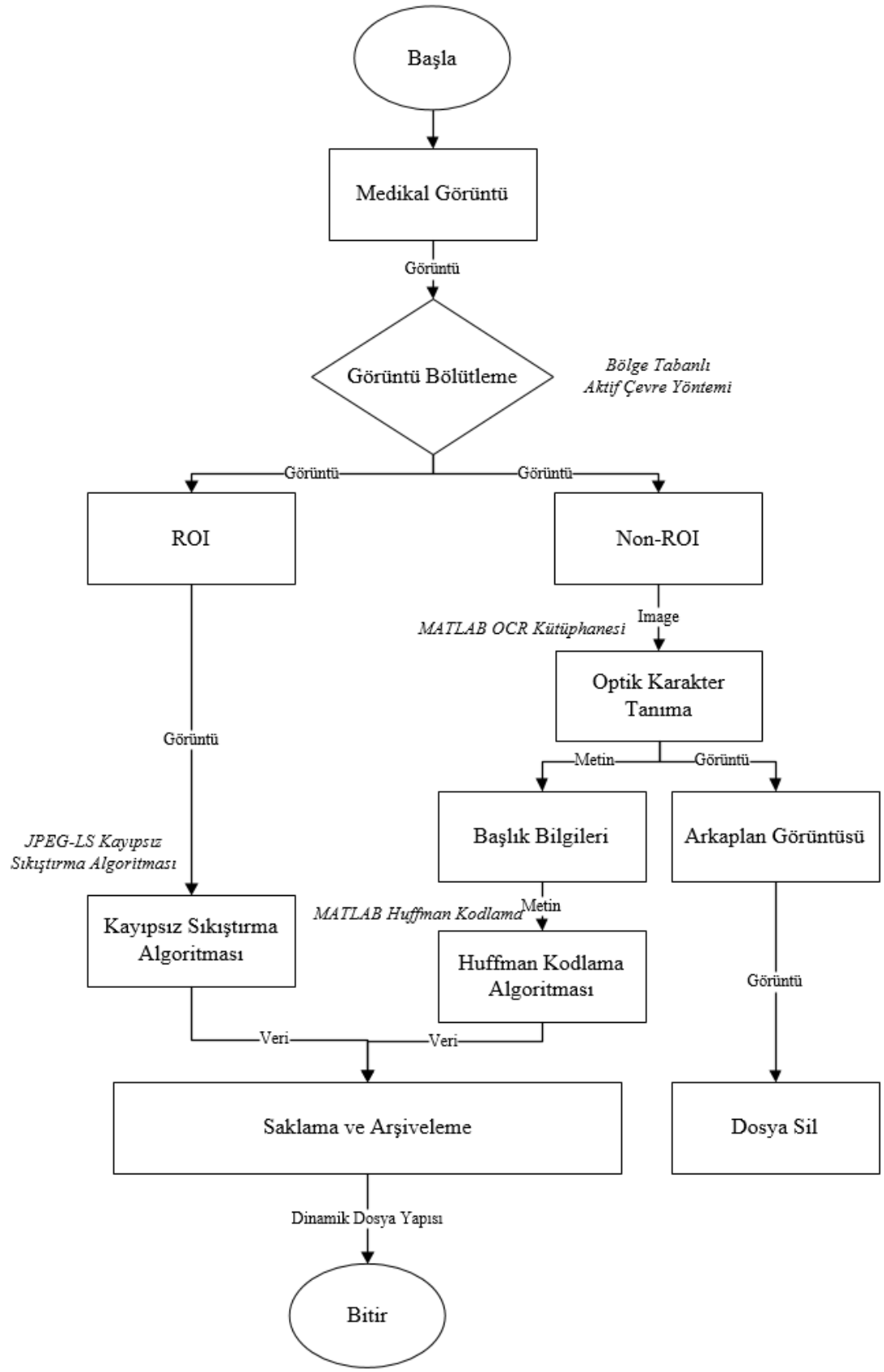
Medikal görüntü sayısı her geçen gün artmaktadır. Bu verilerin boyutu yüksek olup teşhis ve tedavi aşamalarında sıkça kullanılmaktadırlar. Uzun yıllar saklanması, arşivlenmesi gereken bu verilerin depolanması ve farklı yerlere iletilmeleri ciddi

problemler doğurmaktadır. Bu veriler arşivlenme aşamasında sıkıştırılmaktadır ancak yine de büyük boyutlarda veriler meydana gelmektedir.

Medikal görüntülerin sıkıştırılması literatürde sıkça çalışmalar yapılan bir alandır. Bu çalışmanın amacı literatürde bulunan medikal görüntü sıkıştırma yöntemlerine alternatif daha başarılı ve efektif bir alternatif sıkıştırma yöntemi geliştirmektir.

3.3.1. Geliştirilen Yöntem

Bu çalışmada, ROI ve OCR'yi birleştirerek yeni bir tıbbi görüntü sıkıştırma ve arşivleme sistemi önerilmiştir. İlk olarak, görüntü bölütleme işlemi orijinal resmi ROI ve ROI olmayan parçalarına ayırmak için uygulanır. İkincisi, görüntünün meta verileri ve başlık bilgilerini çıkartmak için görüntünün ROI dışı kısmına OCR işlemi uygulanmaktadır. Daha iyi bir sıkıştırma oranı üretmek için verileri ayıklamak için Huffman kodlama algoritması kullanılır. Veriyi elde ettikten sonra, görüntünün herhangi bir bilgi içermeyen non-ROI bölümü atılır. Bundan sonra, herhangi bir veri kaybını önlemek için ROI alanına kayıpsız sıkıştırma yöntemi uygulanır. Son olarak, sıkıştırılmış ROI verileri ve çıkartılan tüm veriler özel olarak tasarlanmış dinamik dosya yapısında saklanır. Böylece etkili ve güçlü büyük tıbbi görüntü saklama ve arşivleme tekniği geliştirilmiştir. Bu çalışmada önerilen model, medikal görüntü arşivleme problemi için ortalama %83,80 oranında daha iyi ve etkili bir yöntemle literatürdeki boşluğu kapatmaktır.
Önerilen görüntü sıkıştırma yöntemi, arşivleme ve veri kodlama akış şeması Şekil 3.4. Geliştirilen yöntem akış şemasında gösterilmektedir.



Şekil 3.4. Geliştirilen yöntem akış şeması

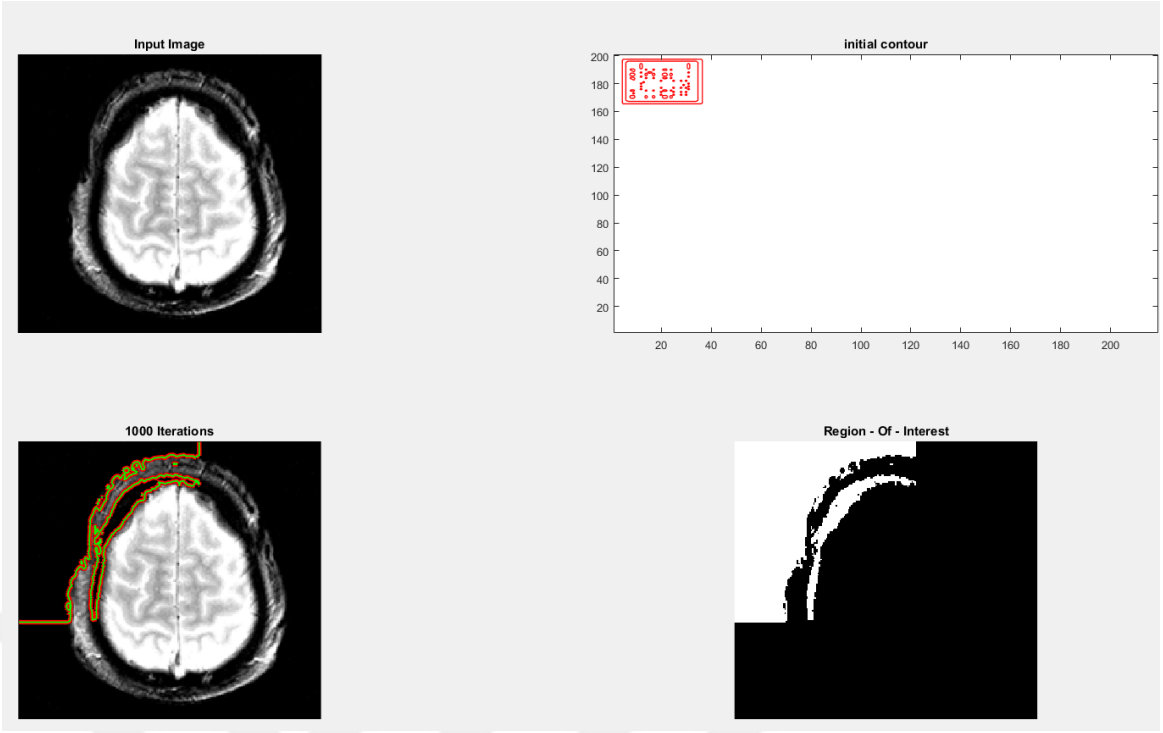
3.3.2. Görüntü Parçalama İşlemi

Birincisi, orijinal görüntü iki bölgeye ayrılmıştır: ROI kısmı ve non-ROI kısmı. Bu süreç için geliştirilen birçok yöntem ve algoritma bulunmaktadır. Aktif çevre yöntemi ve Chan-Vese aktif kontur yöntemi bu alandaki yaygın algoritmalardır [59]. Bölge tabanlı aktif çevre yöntemi, Chan-Vese aktif kontur yönteminden daha verimli olup daha keskin sonuçlara sahiptir [60]. Chan-Vese aktif çevre yöntemi ile bölge tabanlı aktif çevre yöntemine kıyaslaması Çizelge 3.2. Görüntü bölütleme karşılaştırmasında görülmektedir [61].

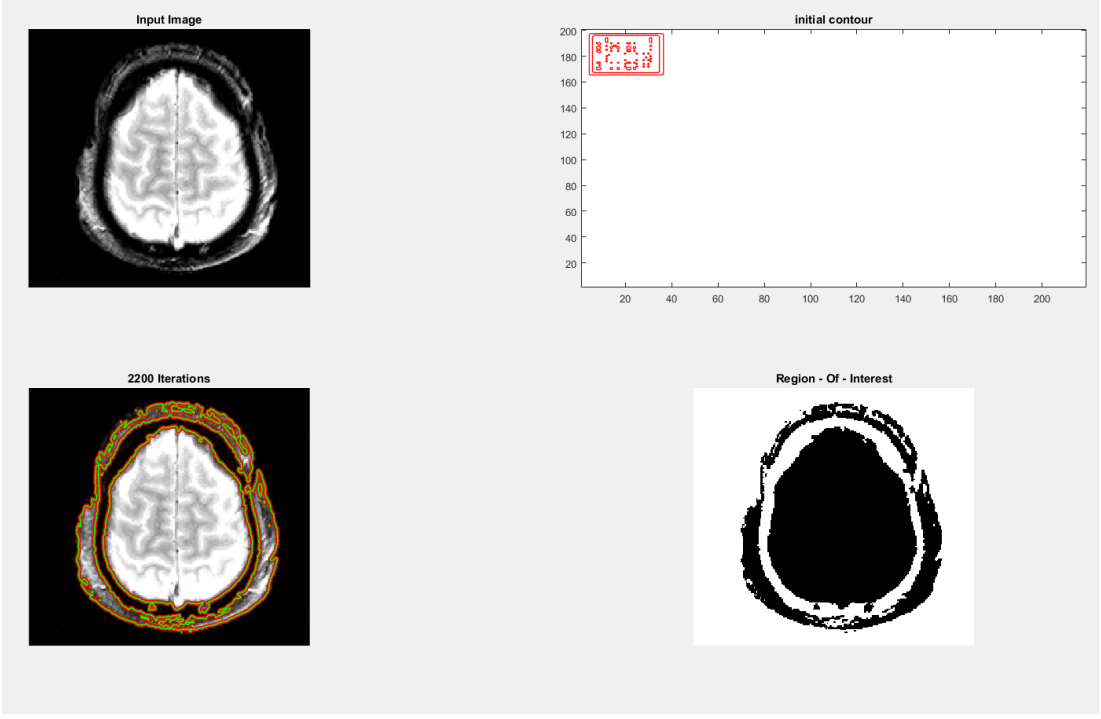
Çizelge 3.2. Görüntü bölütleme karşılaştırması

Chan-Vese aktif kontur yöntemi			Bölge tabanlı aktif çevre yöntemi		
Ortalama	Medyan	Standart Sapma	Ortalama	Medyan	Standart Sapma
9,77	9,72	5,50	7,67	7,70	4,32

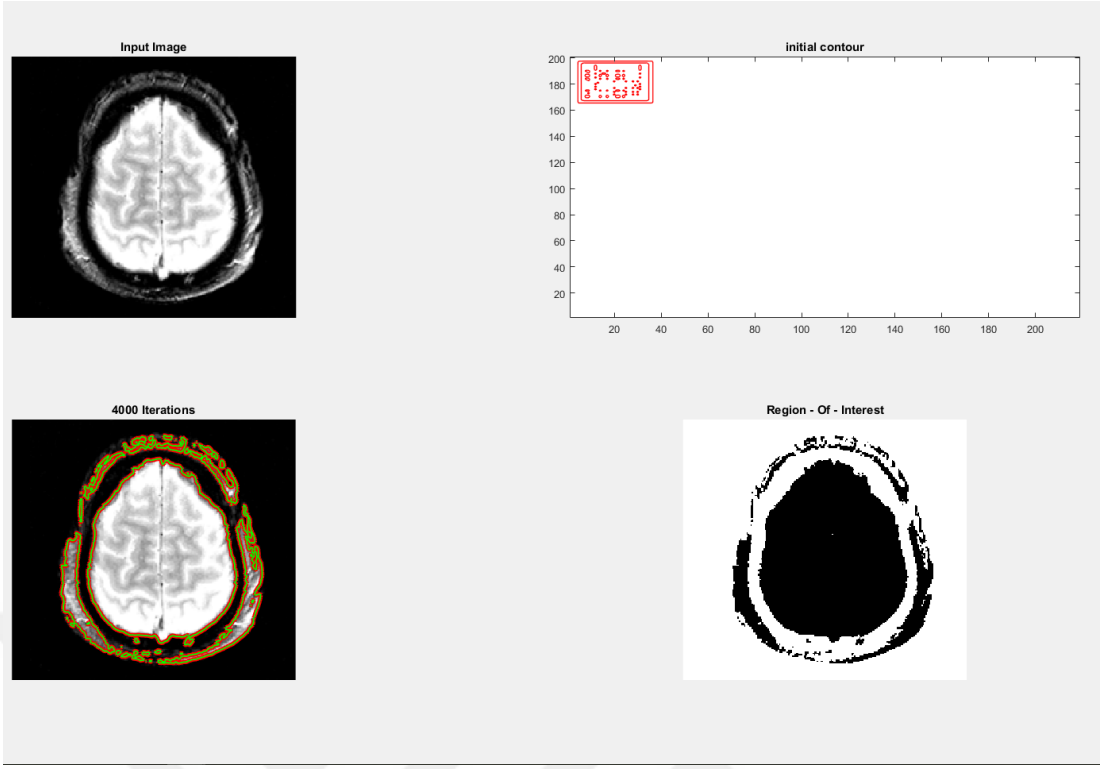
Çalışma sırasında öncelikle Chan-Vese aktif kontrol yöntemi denenmiş ve tekrar sayılarına göre Şekil 3.5. Chan-Vese aktif kontur yöntemi 1000 tekrarlanma, Şekil 3.6. Chan-Vese aktif kontur yöntemi 2200 tekrarlanma ve Şekil 3.7. Chan-Vese aktif kontur yöntemi 4000 tekrarlanma sonuçlar elde edilmiştir. Ayrıca Chan-Vese algoritması için kullanılan kod bloğu EK 1EK 1'de sunulmuştur [62].



Şekil 3.5. Chan-Vese aktif kontur yöntemi 1000 tekrarlanma



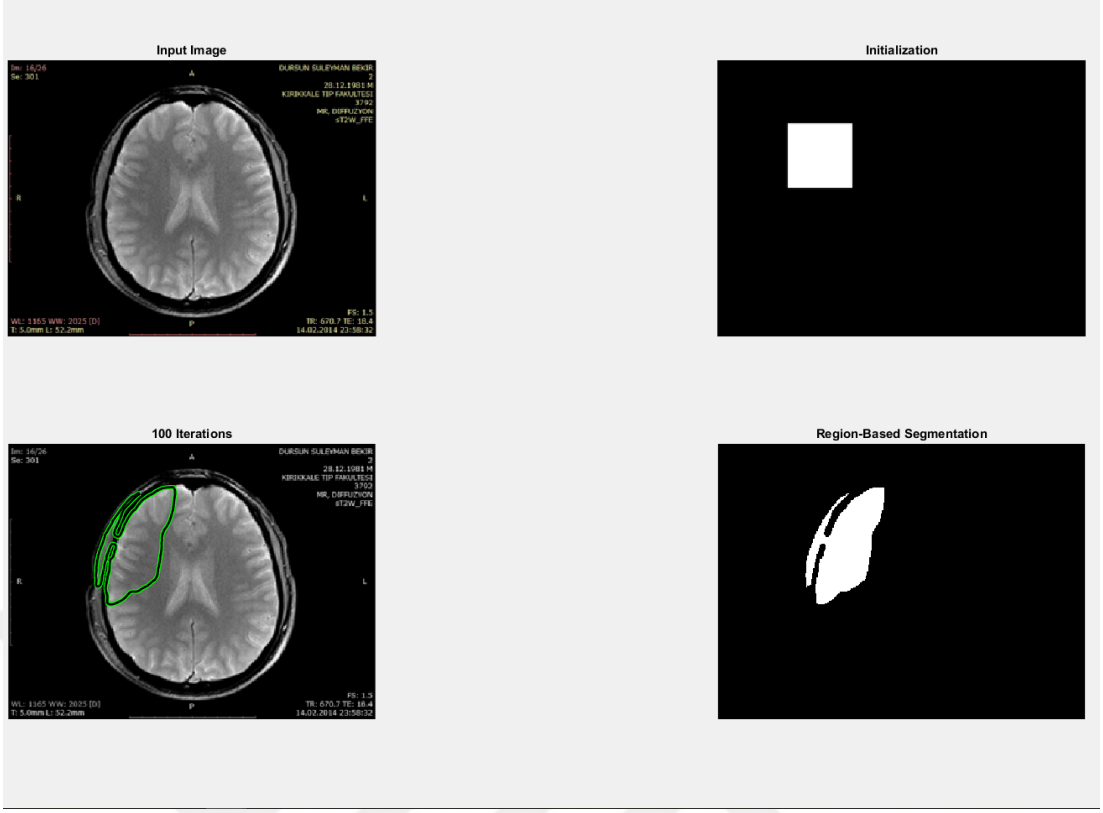
Şekil 3.6. Chan-Vese aktif kontur yöntemi 2200 tekrarlanma



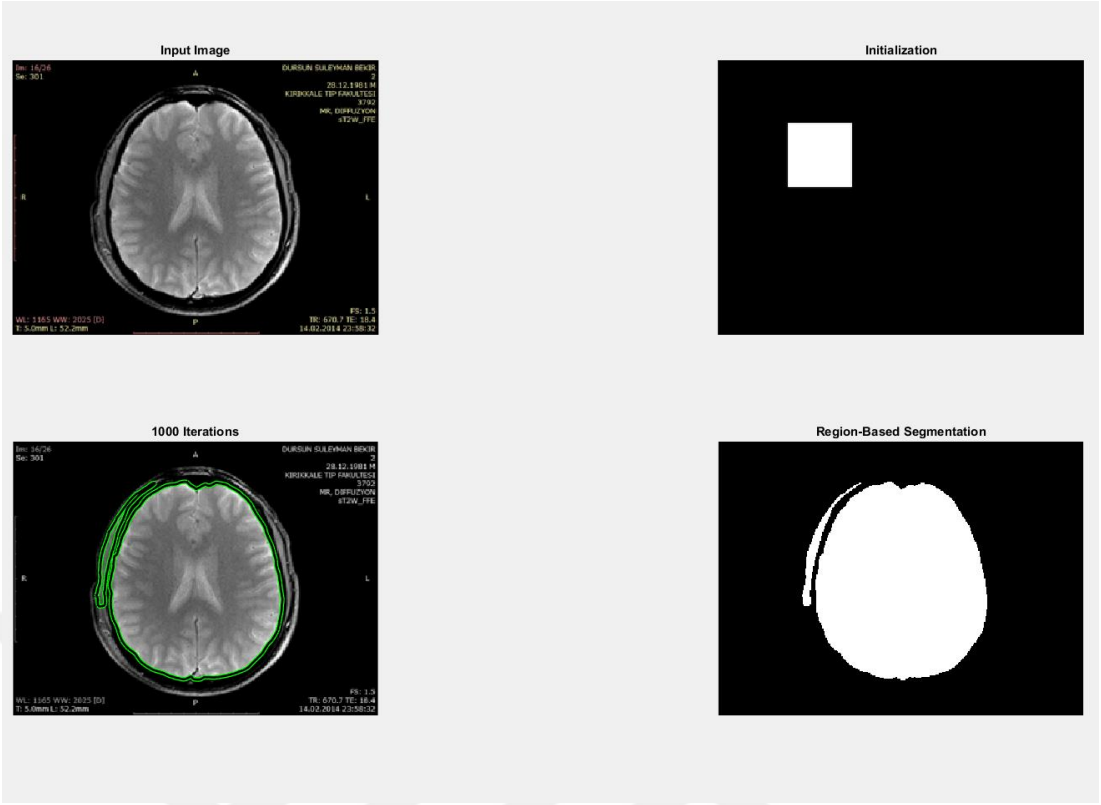
Şekil 3.7. Chan-Vese aktif kontur yöntemi 4000 tekrarlanma

Chan-Vese algoritmasında parametre olarak geçilen tekrarlanma (iteration) sayısı artırıldıkça işlem sonucunda elde edilen doğruluk oranı artmaktadır. Ancak tekrarlanma sayısının artırılması çok büyük performans ve zaman kaybına neden olmaktadır.

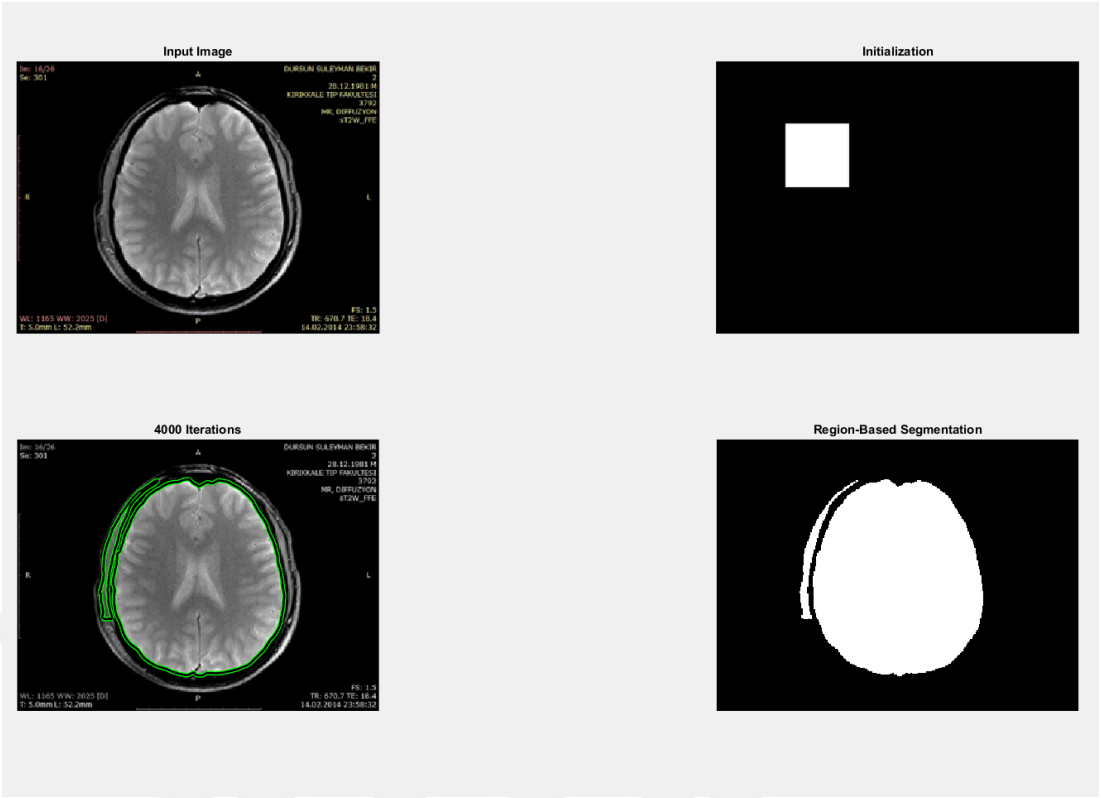
Çalışmada değerlendirilen bir başka yöntem ise bölge tabanlı aktif kontur yöntemidir. Chan-Vese algoritmasına göre çok daha az tekrarlanma sayısına ve zamana ihtiyaç duyan bir yöntemdir. Bölge tabanlı aktif kontur yöntemi de denenmiş ve tekrar sayılarına göre Şekil 3.8. Bölge tabanlı aktif kontur yöntemi 100 tekrarlanma, Şekil 3.9. Bölge tabanlı aktif kontur yöntemi 1000 tekrarlanma ve Şekil 3.10. Bölge tabanlı aktif kontur yöntemi 4000 tekrarlanma sonuçlar elde edilmiştir. Ayrıca bölge tabanlı aktif kontur yöntemi için kullanılan kod bloğu EK 2'de sunulmuştur [63].



Şekil 3.8. Bölge tabanlı aktif kontur yöntemi 100 tekrarlanma

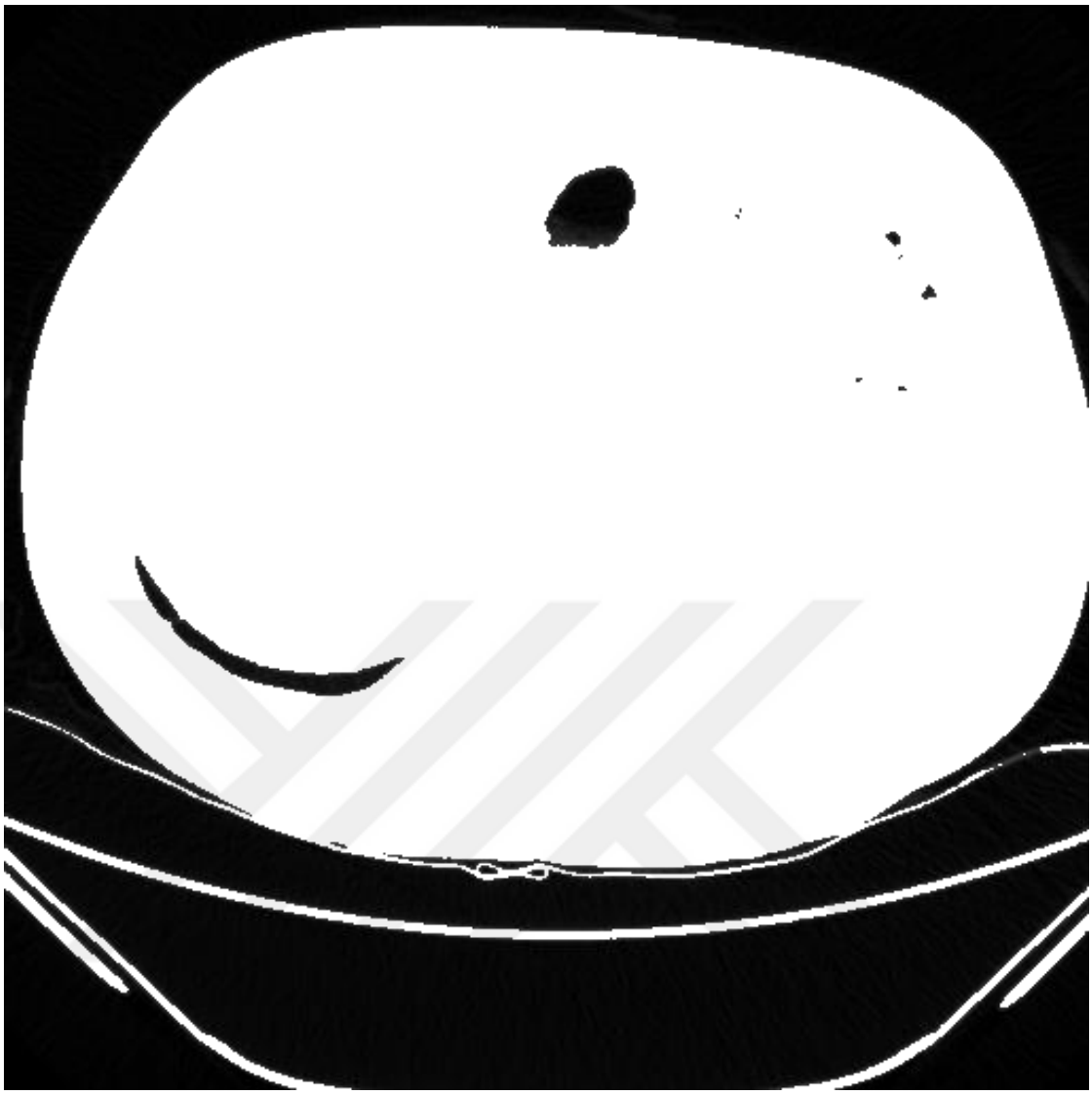


Şekil 3.9. Bölge tabanlı aktif kontur yöntemi 1000 tekrarlanma

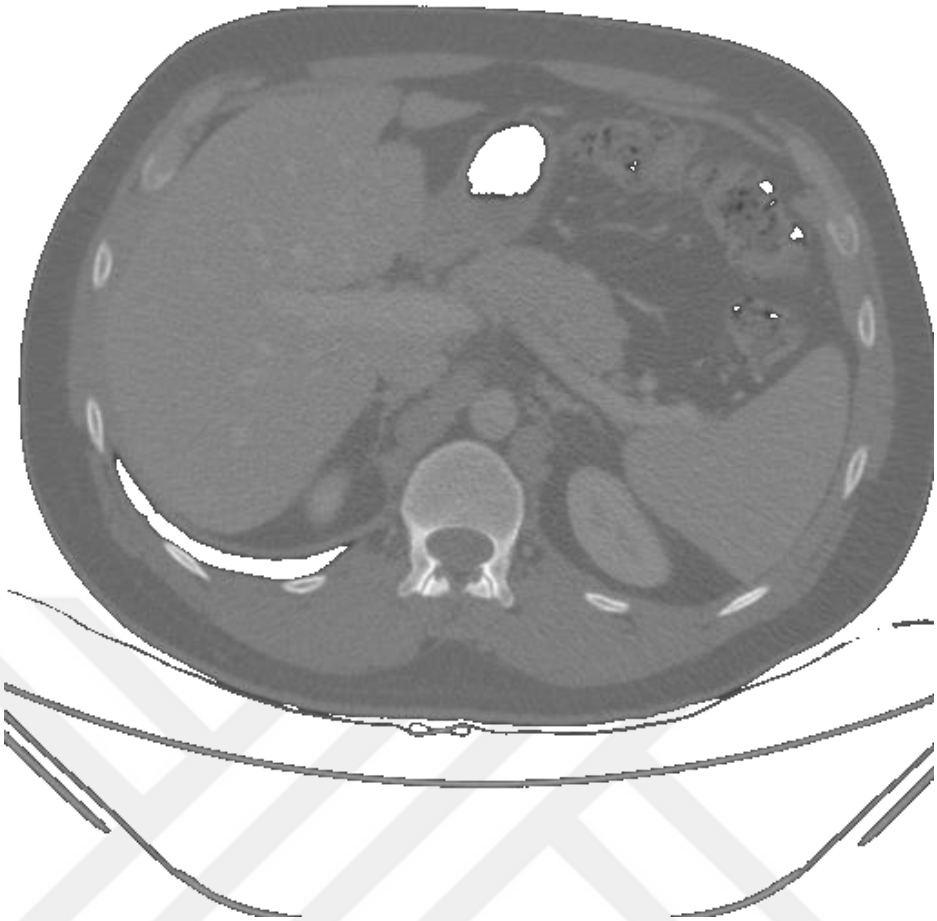


Şekil 3.10. Bölge tabanlı aktif kontur yöntemi 4000 tekrarlanma

Bu iki yöntemin yanı sıra Otsu eşikleme yöntemi de çalışmada görüntü parçalama işlemi için test edilmiştir. Bu işlem sonucu elde edilen Şekil 3.11. Otsu eşikleme yöntemi non-ROI görüntüsü ve Şekil 3.12. Otsu eşikleme yöntemi ROI bölgesi sonuçları gibidir. Kullanılan kod bloğu EK 3'te sunulmaktadır [64, 65].



Şekil 3.11. Otsu eşikleme yöntemi non-ROI görüntüsü



Şekil 3.12. Otsu eşikleme yöntemi ROI bölgesi

Böylece, bölge tabanlı aktif kontur yöntemi, ROI ve ROI olmayan kısımları ayırmak için medikal görüntüye uygulanmıştır.

3.3.3. ROI Bölgesi İşlemleri

Görüntünün ROI kısmının önemi göz önünde bulundurulduğunda, orijinal resmin bu bölümüne kayıpsız görüntü sıkıştırma algoritmaları uygulanmıştır. Düşük hesaplama karmaşıklığına sahip olan ve iyi bilinen bir kayıpsız JPEG sıkıştırma algoritması olan JPEG-LS, giriş/çıkış işlem adımlarını ve çalışma zamanını azaltan bir sıkıştırma algoritması türüdür [66]. Geliştirilen yaklaşımın temel fikri, görüntünün yalnızca ROI bölümünün JPEG-LS kayıpsız sıkıştırma yöntemi ile sıkıştırılmış olmasıdır ancak

kayıpsız sıkıştırılan bu alan Non-ROI bölgesine daha düşük sıkıştırma oranı sağlamaktadır.

Kullanılan JPEG-LS algoritmasına ait kod EK 4'te gösterilmektedir [67].

3.3.4. ROI Dışı Bölgesi İşlemleri

Bu yaklaşımın ana katkılarından biri, medikal görüntünün Non-ROI bölümünde bulunan tanımlayıcı bilgileri OCR ile çıkarmaktır. OCR işlemi öncelikle Şekil 3.13. OCR metin veri tespitinde ve Şekil 3.14. OCR metin veri tespit aşamaları görüldüğü gibi görüntü üzerinde alanları tespit etmektedir.



Şekil 3.13. OCR metin veri tespiti



Şekil 3.14. OCR metin veri tespit aşamaları

OCR işlemi sonucu elde edilen veriler Çizelge 3.3. OCR işlem sonucunda görülmektedir. Kullanılan kod bloğu EK 5'te gösterilmektedir [68].

Çizelge 3.3. OCR işlem sonucu

Sıra	Veri	Koordinat				Görüntü
1	DURSUN	1617	13	91	16	Test görüntüsü (a)
2	SULEYMAN	1719	13	118	16	
3	BEKIR	1849	13	67	16	
4	2	1903	41	12	16	
5	28.12.1981	1762	69	125	16	
6	M	1899	69	15	16	
7	KIRIKKALE	1623	97	118	16	
8	TIP	1750	97	37	16	
9	FAKULTESI	1797	97	117	16	
10	3792	1862	125	53	16	
11	MR,	1737	153	38	20	
12	DIFFUZYON	1787	153	127	16	
13	sT2W_FFE	1802	181	113	19	
14	Im:	8	13	36	16	
15	1/26	58	12	48	20	
16	Se:	7	41	34	16	
17	301	54	41	38	16	
18	FS:	1836	1000	33	16	
19	1.5	1883	1000	32	16	
20	WL:	7	1028	40	16	
21	769	60	1028	39	16	
22	WW:	109	1028	49	16	
23	1336	172	1028	52	16	
24	[D]	235	1027	33	21	
25	TR:	1701	1028	36	16	
26	670.7	1749	1028	62	16	
27	TE:	1820	1028	35	16	
28	18.4	1869	1028	46	16	
29	T:	6	1056	19	16	
30	5.0mm	38	1056	75	16	
31	L:	124	1056	17	16	
32	-37.8mm	153	1056	100	16	
33	14.02.2014	1679	1056	124	16	
34	23:58:32	1813	1056	102	16	

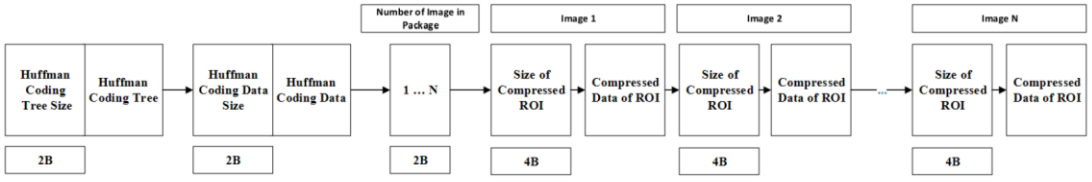
OCR işlemi sayesinde görüntünün meta verileri ve başlık bilgileri elde edilmektedir. Konum bilgisi de orijinal görüntünün yeniden yapılandırılması için kaydedilir. Gerekli bilgiler aldıktan sonra, ROI dışı bölümün geri kalanı gereksizdir ve yok sayılır. Bu yaklaşım, çalışmayı literatürde bulunan diğer çalışmalardan farklı kılan bir diğer

önemli noktadır. Literatürdeki diğer yöntemler Non-ROI bölgesine görüntü verilerini depolar ki bu işlem arşivleme sistemlerinin etkinliğini önemli derecede azaltmaktadır.

3.3.5. Saklama ve Arşivleme İşlemleri

Son işlem depolama ve arşivleme işlemidir. Özel ve dinamik bir dosya yapısı aşağıda gösterildiği gibi tasarlanmıştır. Buna göre, çıktı dosyası dört veri yapısı bölümünden oluşur; Huffman kodlama ağacı boyutu (2B) ve veriler, Huffman kodlama veri boyutu (2B) ve veriler, pakette (2B) resim sayısı ve sıkıştırılmış ROI boyutu (4B) ve veriler. İlk yapı, Huffman ağacının boyut ve veri kısmıdır. Bu ağaç, görüntünün non-ROI bölümünün meta verilerinin anahtarlarını içerir. Paketin ikinci kısmı Huffman veri boyutunu ve verilerini depolar. Paketin üçüncü kısmı paketteki toplam görüntü sayısını depolar. Son olarak, paketin son kısmı görüntünün sıkıştırılmış ROI bölümünün boyutunu saklar ve bu veriler resimlerin sayısı kadardır. Boyut bilgisini depolayan veri üstbilgisi bilgileri, paketlerdeki düzensizliği önlemek için veri yapılarını yorumlamada önemli bir role sahiptir.

Ayrıca, tıbbi bir görüntüleme paketi sadece bir seri veya birden fazla görüntü serisi içerebilir. Sistem, bir Huffman şifreleme ağacı ve görüntü bilgileri sayısı olan veri bilgisi içeren bir dinamik dosya oluşturur. Bu dinamik dosya yapısında bir diğer önemli nokta, her görüntünün meta veri bilgisine sahip olması ve Huffman veri yapılarında saklanmasıdır.



Şekil 3.15. Dinamik dosya yapısı

Bu yapı, boyutu (1 KB) tanımlanmış bir dizi çıkarılan veri kaydı içerir. Bu yapıdan sonra, ROI boyutu ve veriyi paketteki görüntü sayısına göre saklar. Paketin bu dinamik dosya yapısı şekilde gösterildiği gibi her resim için özyinelemelidir. Veri başlıkları, dosya sisteminin tutarlılığı, kararlılığı ve doğruluğu için tanımlanır.

Kullanılan Huffman kod bloğu EK 6'da gösterilmektedir [69].

3.4. Bulut Tabanlı Sıkıştırma Problem Tanımı

Geliştirilen sıkıştırma yöntemi tek düğüm üzerinde çalışmaktadır. Bu işlemlerin daha hızlı ve efektif olarak gerçekleştirilebilmesi için aynı sistem bulut tabanlı olarak geliştirilmiştir. Bu sayede arşiv sistemi otomatik yedeklenebilen, genişleyebilen ve hata toleransına sahip performanslı bir sistem haline getirilmiştir.

3.4.1. Geliştirilen Yöntem

Bu çalışmada ise, tek düğüm sıkıştırma probleminde kullanılan mimari çözüm Hadoop ve MongoDB mimarisine dayalı olarak geliştirilmiştir. Geliştirilen platform dört düğümlü bir Hadoop kümesinden, bir isim düğümünden ve üç veri düğümünden oluşur. Dağıtık MapReduce algoritması ve MongoDB her düğümde çalışmaktadır. Platform dört ana aşamadan oluşmaktadır. Birincisi, tıbbi görüntü paketi bir veya daha fazla görüntü içermektedir. Bu paketteki her görüntü, paralel işlem için Hadoop kümesindeki uygun sunucuya gönderilir. Tüm işlemler MapReduce programlama modeliyle Hadoop kümesinde yürütülür. İkinci aşamada, ham veriler ilk problemde kullanılan yöntemler ki çalışmanın bundan sonrasında medikal görüntü işleme paketi olarak adlandırılacaktır, kullanılarak işlenir ve sıkıştırılır. Bu paketin son aşamasında, elde edilen veriler ve karşılık gelen anahtarlar MongoDB'de saklanır. Platformdaki son özellik bir kullanıcı ara yüzüne sahip bir arama motorudur. Gelişmiş arama motoru, hasta kimliği veya vatandaşlık kimlik numarası, adı ve soyadı gibi tanımlanmış

kriterleri aramak için tasarlanmıştır. Arama tamamlandıktan sonra, sistemdeki tüm aşamalar ters çevrilir ve görüntü kullanıcıya teslim edilir.

3.4.2. Hadoop Ön İşleme Süreci

Tek düğüm üzerinde yapılan tüm işlemlerin bulut tabanlı sıkıştırma aşamasında da yapılmaları gerekmektedir. Bu aşamalara ait detaylar aşağıdaki gibidir.

3.4.2.1. Optik Karakter Tanıma

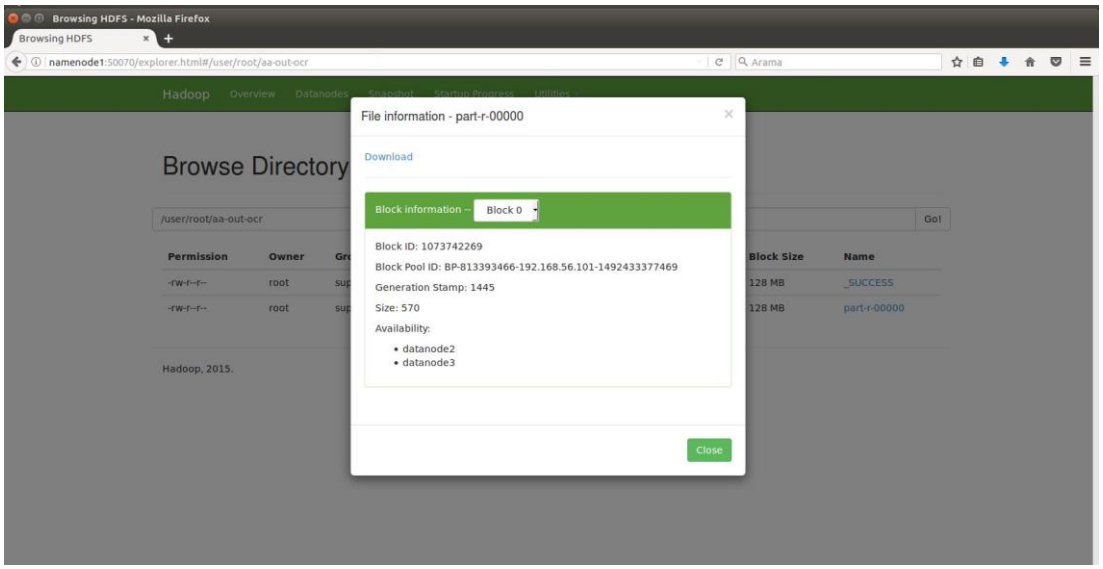
Tek düğüm üzerinde görüntünün non-ROI bölgesine uygulanan ve görüntü üzerindeki tüm karakterlerin elde edilmesini sağlayan optik karakter tanıma bu yöntemde de uygulanmaktadır. Kullanılan optik karakter tanıma kod bloğu EK 7'de gösterilmektedir.

OCR işleminin konsol ekranında başlatılması Şekil 3.16. Hadoop optik karakter tanıma başlangıcında görülmektedir.

```
root@namenode1:~# hadoop jar /home/erdal/workspace/OCR/mr2/target/ocrmr-*-with-dependencies.jar ocr.mr.OCRMR /tmp/PACS
/aa.png aa-out-ocr -Djava.library.path=/usr/lib/jni
17/05/24 13:54:16 INFO client.RMProxy: Connecting to ResourceManager at namenode1/192.168.56.101:8032
17/05/24 13:54:19 INFO input.FileInputFormat: Total input paths to process : 1
17/05/24 13:54:19 INFO input.CombineFileInputFormat: DEBUG: Terminated node allocation with : CompletedNodes: 2, size
left: 787648
17/05/24 13:54:19 INFO mapreduce.JobSubmitter: number of splits:1
17/05/24 13:54:19 INFO mapreduce.JobSubmitter: Submitting tokens for job: job_1495622123680_0007
17/05/24 13:54:20 INFO impl.YarnClientImpl: Submitted application application_1495622123680_0007
17/05/24 13:54:20 INFO mapreduce.Job: The url to track the job: http://namenode1:8088/proxy/application_1495622123680_
0007/
17/05/24 13:54:20 INFO mapreduce.Job: Running job: job_1495622123680_0007
17/05/24 13:54:29 INFO mapreduce.Job: Job job_1495622123680_0007 running in uber mode : false
17/05/24 13:54:29 INFO mapreduce.Job:  map 0% reduce 0%
17/05/24 13:54:37 INFO mapreduce.Job:  map 100% reduce 0%
17/05/24 13:54:44 INFO mapreduce.Job:  map 100% reduce 100%
17/05/24 13:54:45 INFO mapreduce.Job: Job job_1495622123680_0007 completed successfully
17/05/24 13:54:45 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=480
                FILE: Number of bytes written=239687
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=787777
                HDFS: Number of bytes written=570
                HDFS: Number of read operations=7
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=2
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=1
                Other local map tasks=1
                Total time spent by all maps in occupied slots (ms)=5992
                Total time spent by all reduces in occupied slots (ms)=4276
                Total time spent by all map tasks (ms)=5992
                Total time spent by all reduce tasks (ms)=4276
                Total vcore-milliseconds taken by all map tasks=5992
                Total vcore-milliseconds taken by all reduce tasks=4276
                Total megabyte-milliseconds taken by all map tasks=6135808
                Total megabyte-milliseconds taken by all reduce tasks=4378624
        Map-Reduce Framework
                Map input records=1
                Map output records=1
                Map output bytes=572
                Map output materialized bytes=466
                Input split bytes=129
                Combine input records=1
                Combine output records=1
                Reduce input groups=1
                Reduce shuffle bytes=466
                Reduce input records=1
                Reduce output records=1
                Spilled Records=2
                Shuffled Maps =1
                Failed Shuffles=0
                Merged Map outputs=1
                GC time elapsed (ms)=248
                CPU time spent (ms)=3170
                Physical memory (bytes) snapshot=434515968
                Virtual memory (bytes) snapshot=4491919360
                Total committed heap usage (bytes)=353501184
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=0
        File Output Format Counters
                Bytes Written=570
```

Şekil 3.16. Hadoop optik karakter tanıma başlangıcı

Yapılan işlem sonucu elde edilen tüm veriler yine Hadoop dosya sistemi olan HDFS'de saklanılır. Bu işlem farklı makinalarda gerçekleşebilir. İşlem sonucunda dosyaların hangi makinalar üzerine kayıt edildiği Şekil 3.17. Hadoop optik karakter tanıma sonuç dosyası kayıt görüntüsünde olduğu gibi görüntülenebilir.



Şekil 3.17. Hadoop optik karakter tanıma sonuç dosyası kayıt görüntüsü

Ayrıca Hadoop optik karakter tanıma sonuç konsol ekranı Şekil 3.18. Hadoop optik karakter tanıma sonuç konsol ekranında görüldüğü gibidir.

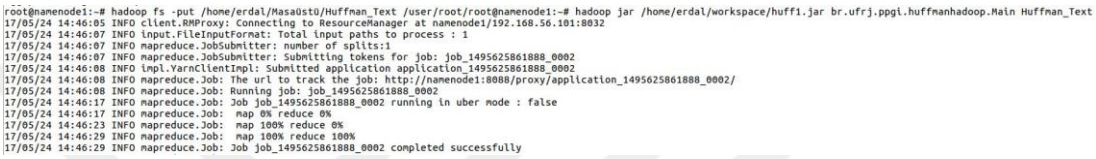
```
root@namenode1:~# hadoop fs -cat /user/root/aa-out-ocr/part-r-00000
aa.png  Im: 1/1 F A'l DURSUN SULEYMAN BEKIR
Se: 3o1 \ 2
28.12.1981 M
4 1 KIRIKKALE TIP FAKULTESI
; b 3792
3 _ _ \ MR, DIFFUZYON
FS: 1.5
WL: 769 ww: 1336 [D] P TR: 670.7 TE: 1s.4
T: 5.omm L: -37.8mm 14.o2.2o14 23:5s:32
```

Şekil 3.18. Hadoop optik karakter tanıma sonuç konsol ekranı

3.4.2.2. Huffman Kodlama Algoritması

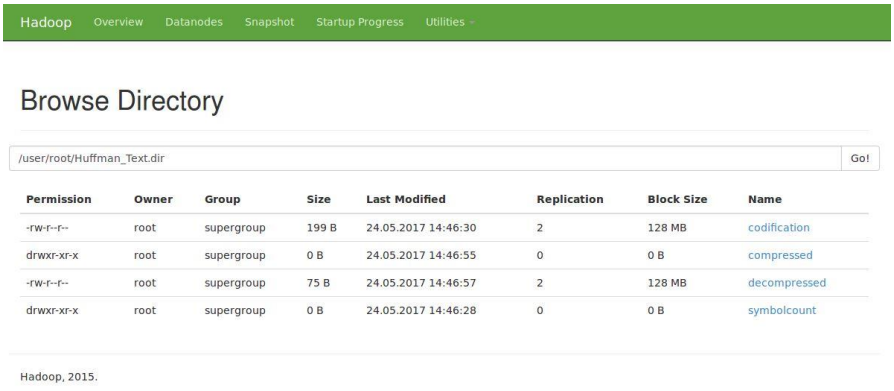
Optik karakter tanıma sonucu elde edilen verilerin daha yüksek oranlarda sıkıştırılmalarını sağlamak için bu verilere Huffman kodlama algoritması uygulanmaktadır. Kullanılan Huffman kodlama algoritması kod bloğu EK 8'de gösterilmektedir.

Huffman kodlama algoritmasının Hadoop üzerinde çağırılması Şekil 3.19. Hadoop huffman kodlama algoritması başlangıçta görüldüğü yapılmaktadır.

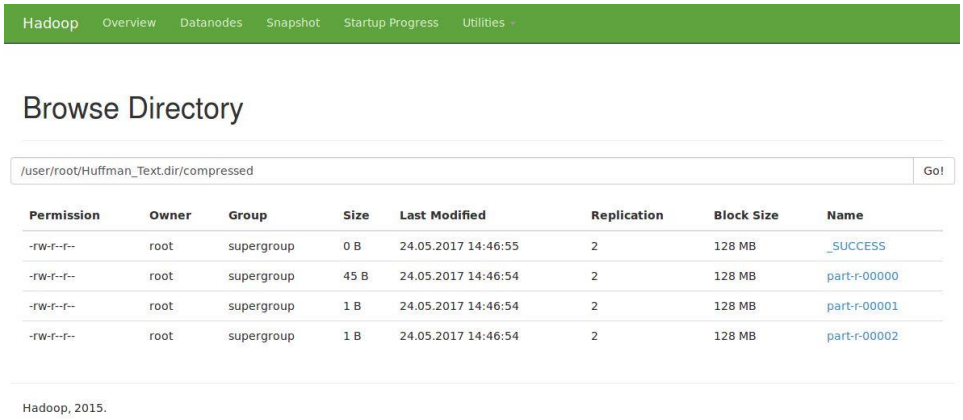


Şekil 3.19. Hadoop huffman kodlama algoritması başlangıç

Bu işlem sonrasında huffman dosyalarının görüntülendiği ekranlar Şekil 3.20. Hadoop huffman dosya görüntüsü, Şekil 3.21. Hadoop huffman sıkıştırma sonuç ekranı ve Şekil 3.22. Hadoop huffman işlem sonucu konsol ekranı gibidir.



Şekil 3.20. Hadoop huffman dosya görüntüsü



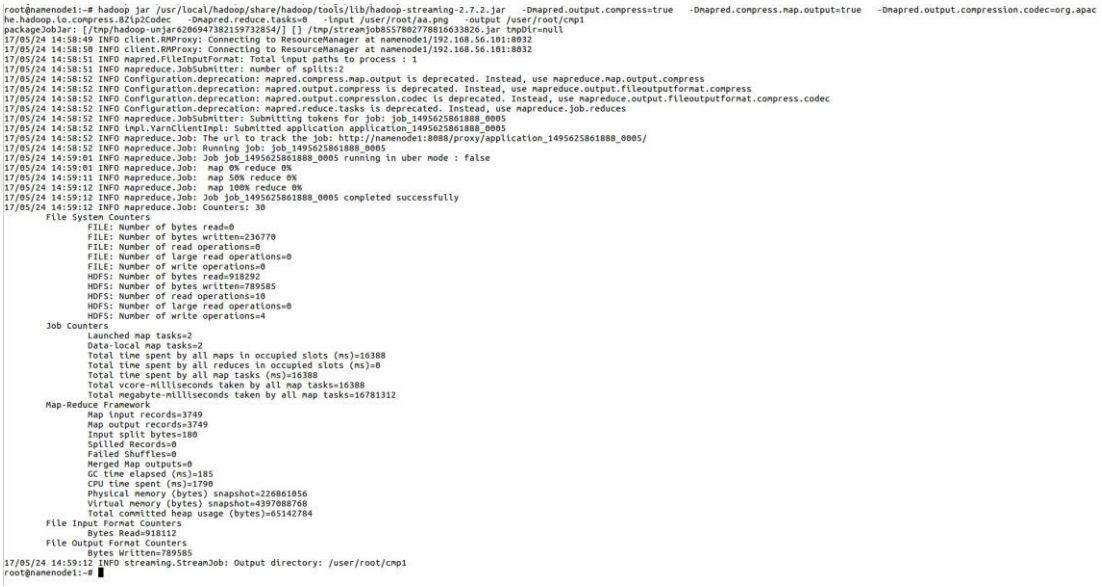
Şekil 3.21. Hadoop huffman sıkıştırma sonuç ekranı

```
17/05/24 14:46:57 INFO mapreduce.Job: Counters: 49
        File System Counters
                FILE: Number of bytes read=247
                FILE: Number of bytes written=474887
                FILE: Number of read operations=0
                FILE: Number of large read operations=0
                FILE: Number of write operations=0
                HDFS: Number of bytes read=574
                HDFS: Number of bytes written=47
                HDFS: Number of read operations=16
                HDFS: Number of large read operations=0
                HDFS: Number of write operations=6
        Job Counters
                Launched map tasks=1
                Launched reduce tasks=3
                Data-local map tasks=1
                Total time spent by all maps in occupied slots (ms)=3536
                Total time spent by all reduces in occupied slots (ms)=13458
                Total time spent by all map tasks (ms)=3536
                Total time spent by all reduce tasks (ms)=13458
                Total vcore-milliseconds taken by all map tasks=3536
                Total vcore-milliseconds taken by all reduce tasks=13458
                Total megabyte-milliseconds taken by all map tasks=3620864
                Total megabyte-milliseconds taken by all reduce tasks=13780992
        Map-Reduce Framework
                Map input records=1
                Map output records=1
                Map output bytes=61
                Map output materialized bytes=205
                Input split bytes=109
                Combine input records=0
                Combine output records=0
                Reduce input groups=1
                Reduce shuffle bytes=205
                Reduce input records=1
                Reduce output records=3
                Spilled Records=2
                Shuffled Maps =3
                Failed Shuffles=0
                Merged Map outputs=3
                GC time elapsed (ms)=289
                CPU time spent (ms)=3090
                Physical memory (bytes) snapshot=558948352
                Virtual memory (bytes) snapshot=8805953536
                Total committed heap usage (bytes)=235147264
        Shuffle Errors
                BAD_ID=0
                CONNECTION=0
                IO_ERROR=0
                WRONG_LENGTH=0
                WRONG_MAP=0
                WRONG_REDUCE=0
        File Input Format Counters
                Bytes Read=80
        File Output Format Counters
                Bytes Written=47
```

Şekil 3.22. Hadoop huffman işlem sonucu konsol ekranı

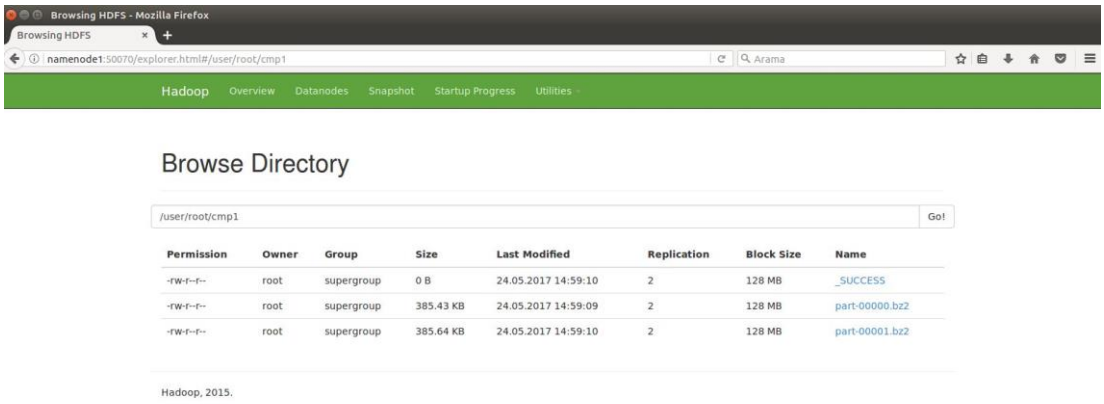
3.4.2.3. Sıkıştırma Algoritması Uygulaması

Görüntünün ROI bölgesine uygulanan kayıpsız sıkıştırma algoritması Hadoop tarafında da uygulanmaktadır. Bu işlemin yapıldığı konsol ekranı Şekil 3.23. Hadoop ROI bölgesi kayıpsız sıkıştırma işlemi konsol ekranında görülmektedir. Kullanılan sıkıştırma algoritması kod bloğu EK 9'de gösterilmektedir.



Şekil 3.23. Hadoop ROI bölgesi kayıpsız sıkıştırma işlemi konsol ekranı

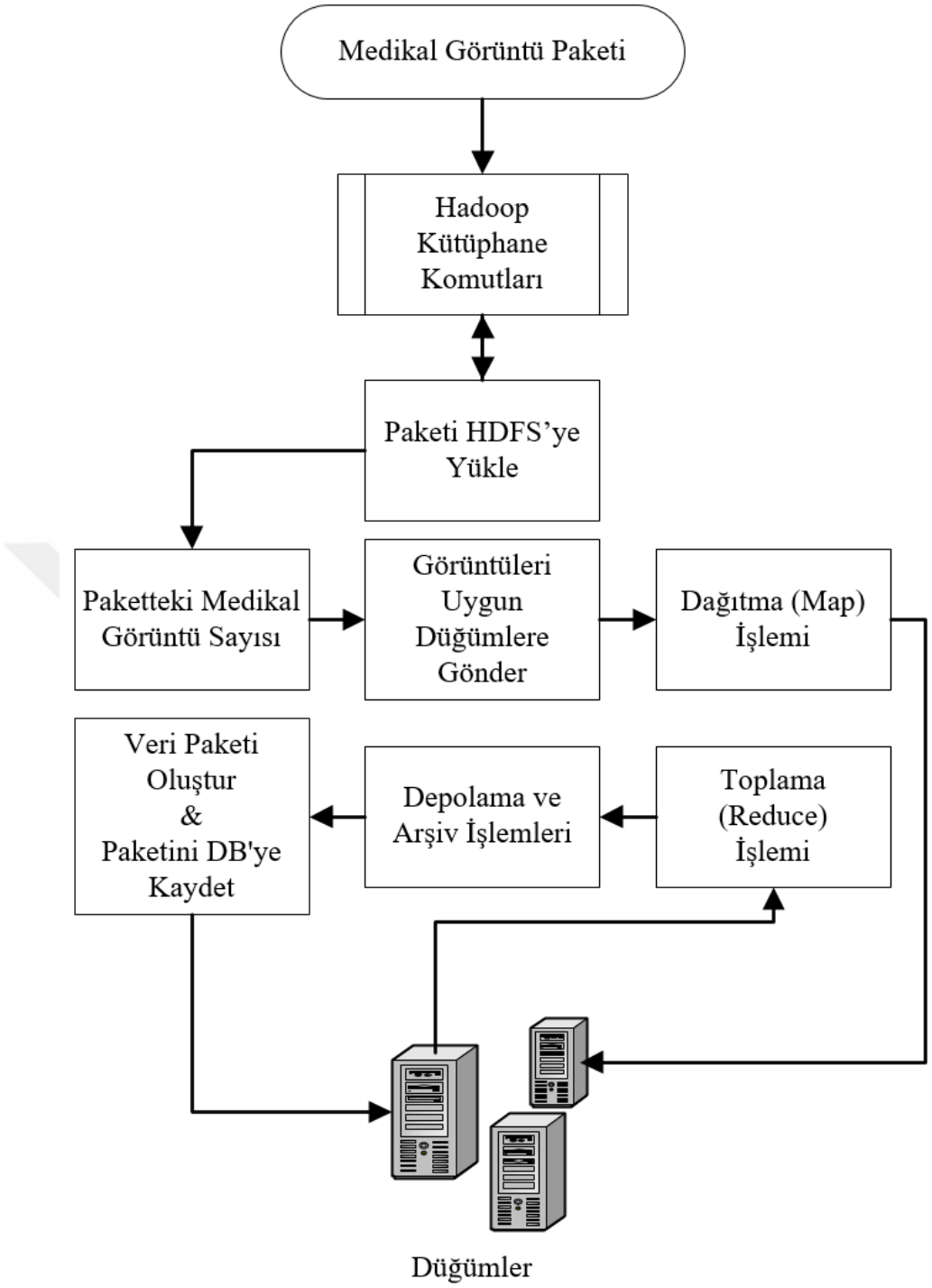
İşlem sonrası Hadoop dosya sistemi HDFS'de oluşan ekran görüntüsü Şekil 3.24. Hadoop ROI bölgesi kayıpsız sıkıştırma işlemi sonuç ekranı gibidir.



Şekil 3.24. Hadoop ROI bölgesi kayıpsız sıkıştırma işlemi sonuç ekranı

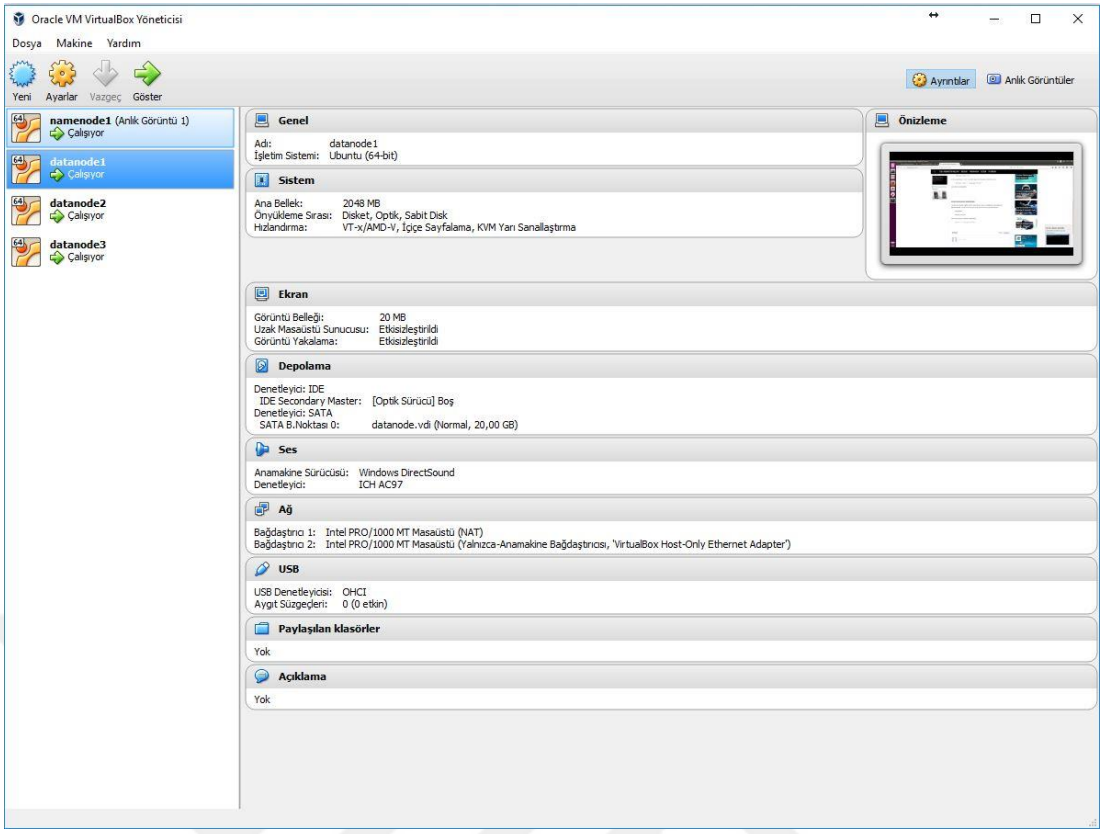
3.4.3. MapReduce/Hadoop Kullanımı

Bu çalışmada, Hadoop kümeleri dağıtık MapReduce algoritmaları çalıştırmak için kullanılmaktadır. Bu küme sayesinde hataya dayanıklı ve paralel çalışan bir tıbbi görüntü işleme platformu oluşturulmuştur. Arşiv sisteminde saklanacak veriler, Hadoop kitaplığı komutları kullanılarak geliştirilen veri arabirimi aracılığıyla HDFS'ye yüklenir. Bu küme önce yüklenen pakette yer alan tıbbi görüntülerin sayısını belirler ve görevleri mevcut her düğüme dağıtır. Dosyayı aldıktan sonra her düğüm, Map işlemini ve çalışmanın önişleme bölümünde açıklanan işlemleri gerçekleştirir. Bu işlem nedeniyle, elde edilen veriler Reduce işlemi ile name adı verilen düğümüne gönderilir. Ardından, pakete ait tüm veriler birleştirilir ve saklanmak için veri tabanına gönderilir. Bu aşamada uygulanan adımlar Şekil 3.25. MapReduce/Hadoop işlem adımlarında gösterilmiştir.



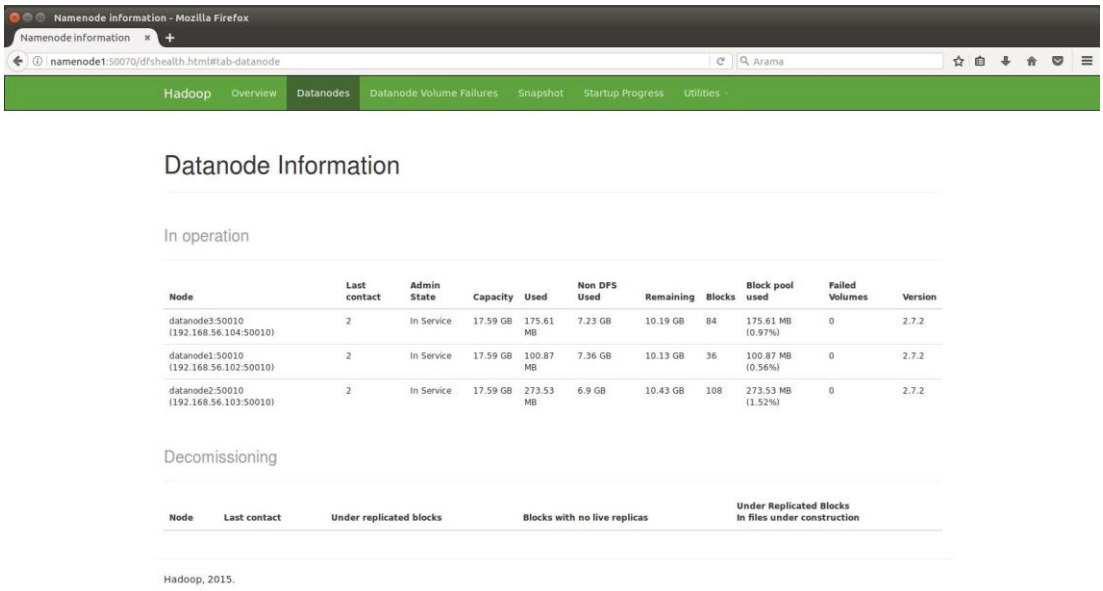
Şekil 3.25. MapReduce/Hadoop işlem adımları

Ayrıca çalışmada kullanılan ortam Oracle VM VirtualBox üzerine kurulmuştur. Kurulum sonrası ekran görüntüsü Şekil 3.26. Oracle vm virtualbox gibidir.



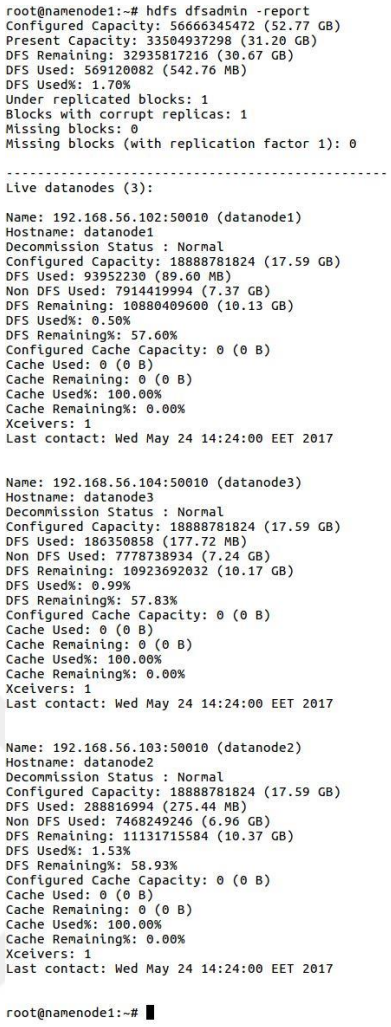
Şekil 3.26. Oracle vm virtualbox

Kurulumu yapılan veri düğümü sanal makinalara ait detayların bulunduğu ekranı Şekil 3.27. Veri düğüm sanal makinaları gibidir.



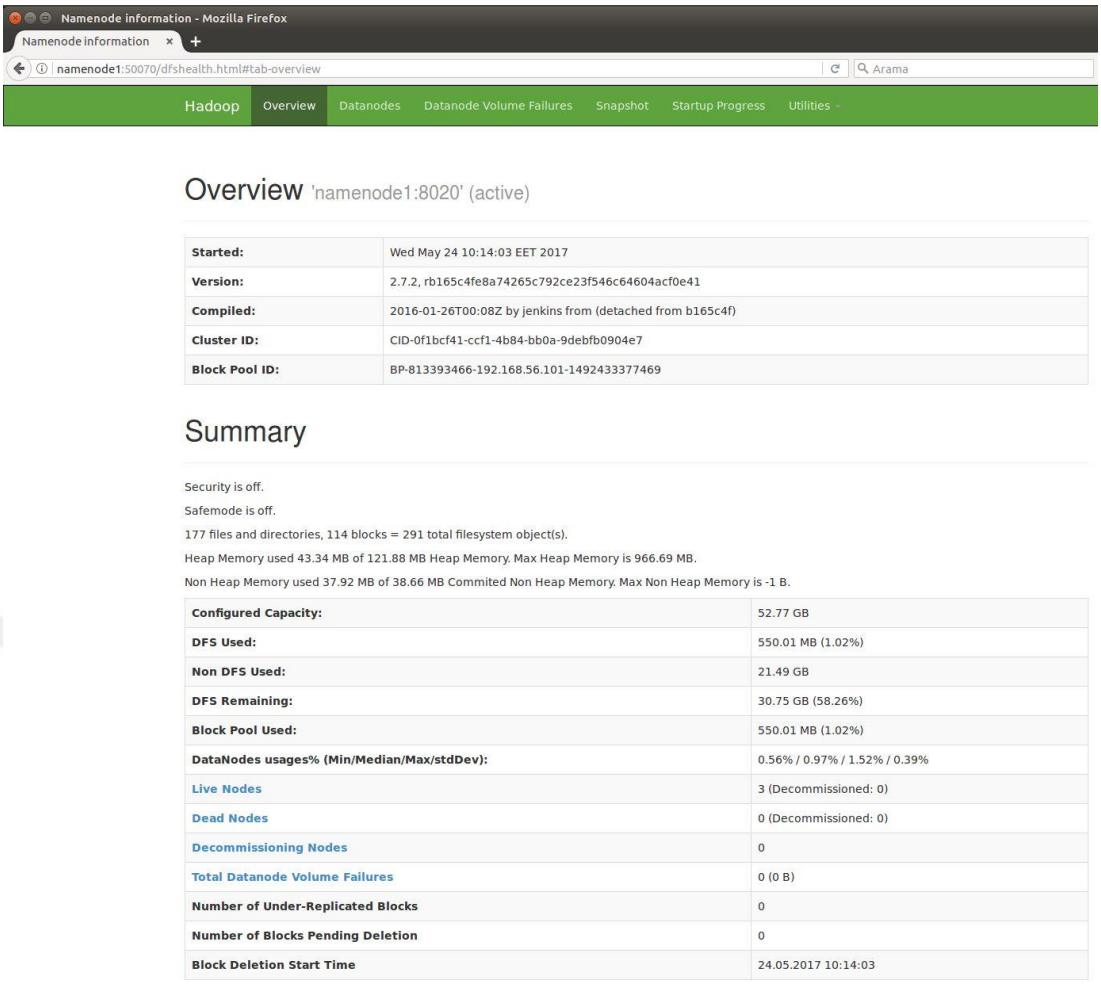
Şekil 3.27. Veri düğüm sanal makinaları

Ayrıca tüm sanal makinalara ait detayların ve raporların bulunduğu konsol ekranı Şekil 3.28. Sanal makina konsol raporları gibidir.



Şekil 3.28. Sanal makina konsol raporları

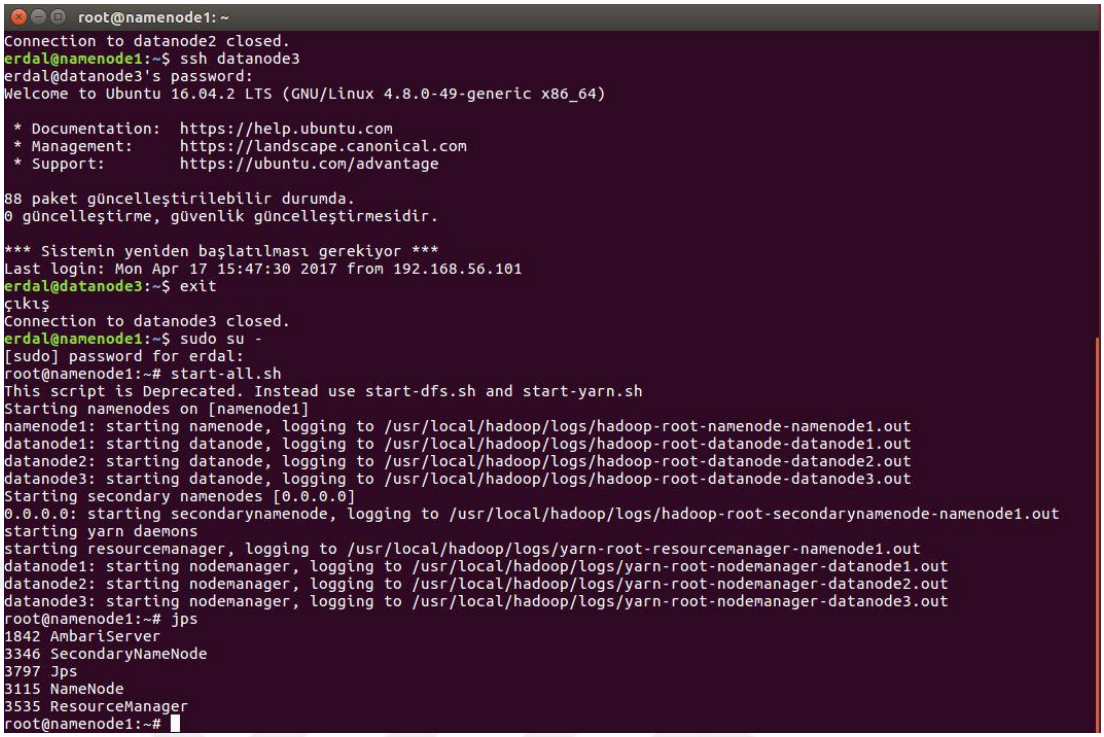
Kurulumu yapılan makinalara görev atamaları tek makine tarafından yapılmaktadır. Bu makine isim düğüm olarak adlandırılır. Bu makineler arasından ana – köle (master – slave) ilişkisi bulunur. Kurulumu yapılan ana makinaya ait bilgi ekranı Şekil 3.29. Ana makina bilgi ekranı gibidir.



Şekil 3.29. Ana makina bilgi ekranı

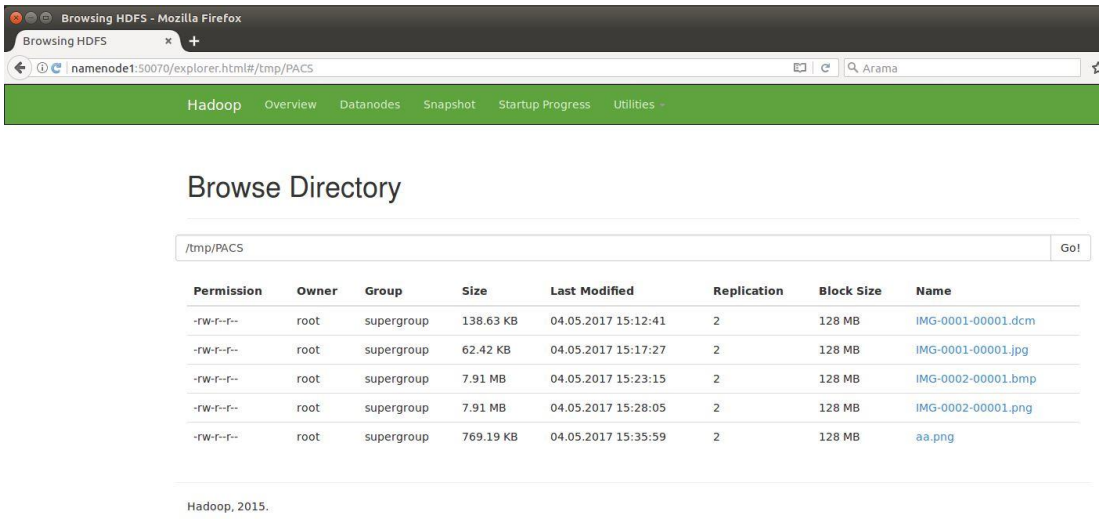
Kurulum sonrasında bu makinalar VM VirtualBox üzerinden çalıştırılır. Sonrasında bu program devreden çıkar. Bu makinaların çalışması Hadoop sunucuların birbirine bağlı ve hazır olarak çalıştıkları anlamına gelmez. Ana makine üzerinden Hadoop servislerinin çalıştırılması ve tüm bağlantıların tamamlandığından emin olunması gerekmektedir. Aksi durumda eksik makine ile küme çalışmaya başlar ve performans olumsuz etkilenir.

Hadoop servislerinin başlatıldığı konsol ekranı Şekil 3.30. Hadoop servis açılışı görülmektedir.



Şekil 3.30. Hadoop servis açılışı

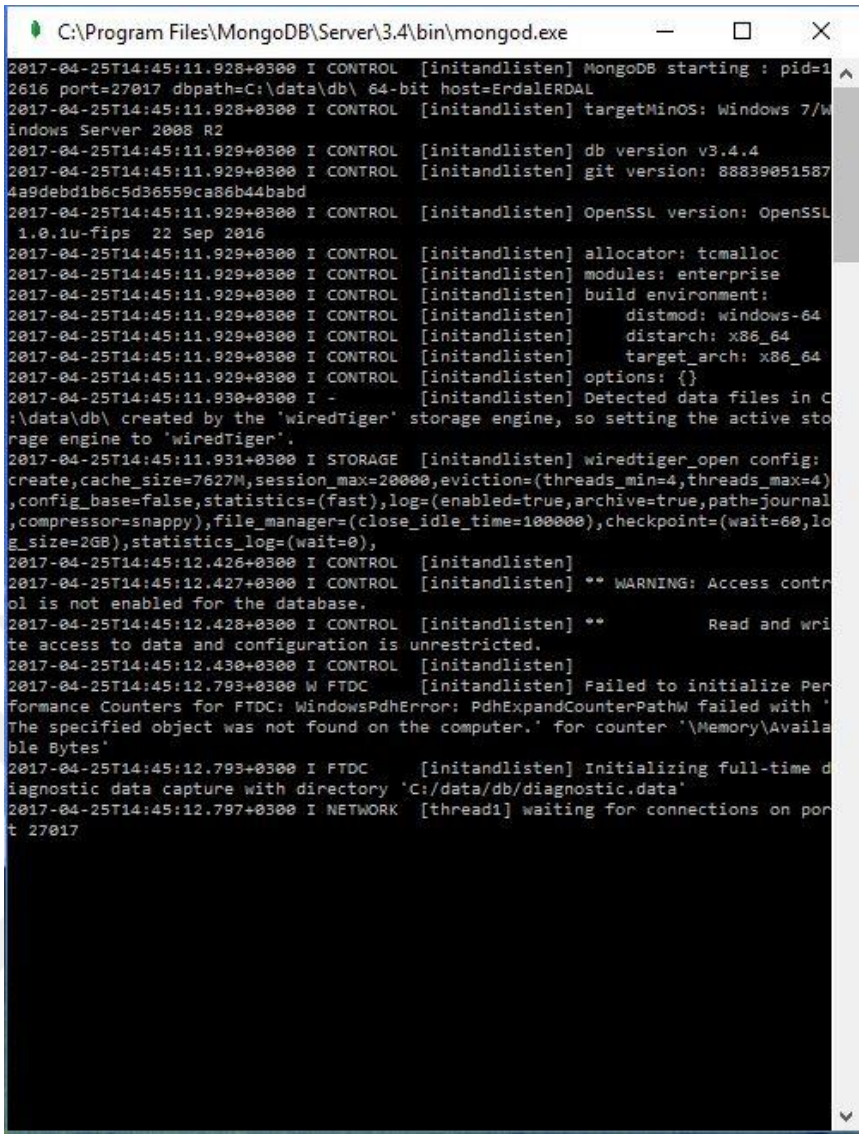
Hadoop servisleri çalışması sonrasında tüm makinalar verilecek görevler için hazır konumda beklemektedir. Ancak bu makinaların ilgili görevleri yapabilmeleri için giriş dosyalarının (input files) Hadoop dosya sistemine yüklenmiş olması gerekmektedir. Yüklenen dosyaların görüntüsü Şekil 3.31. Hadoop dosya sistemi gibidir.



Şekil 3.31. Hadoop dosya sistemi

3.4.4. MongoDB ve GridFS Kullanımı

Dosya paketi, MongoDB'de depolama ve arşivleme işlemi nedeniyle oluşturulmuştur. MongoDB yerleşik fonksiyonları ile veri koleksiyonlarını otomatik olarak parçalayabilir ve bölünmüş verileri mevcut sunucularda depolayabilir. Bu şekilde, düğümler arasında dengeli bir yük dağılımı dinamik olarak sağlanır. Tüm bu özellikleri ile MongoDB'deki kapasite ve verimlilik, düğüm sayısını artırarak artırılabilir. Hataya dayanıklı sistem dört düğümden oluşur ve veriler bu düğümlerde saklanır. MongoDB kurulumu sonrasında elde edilen görüntü Şekil 3.32. MongoDB bağlantı bekliyor ekranında görüldüğü gibidir. Kullanılan MongoDB kod bloğu EK 10'da gösterilmektedir.

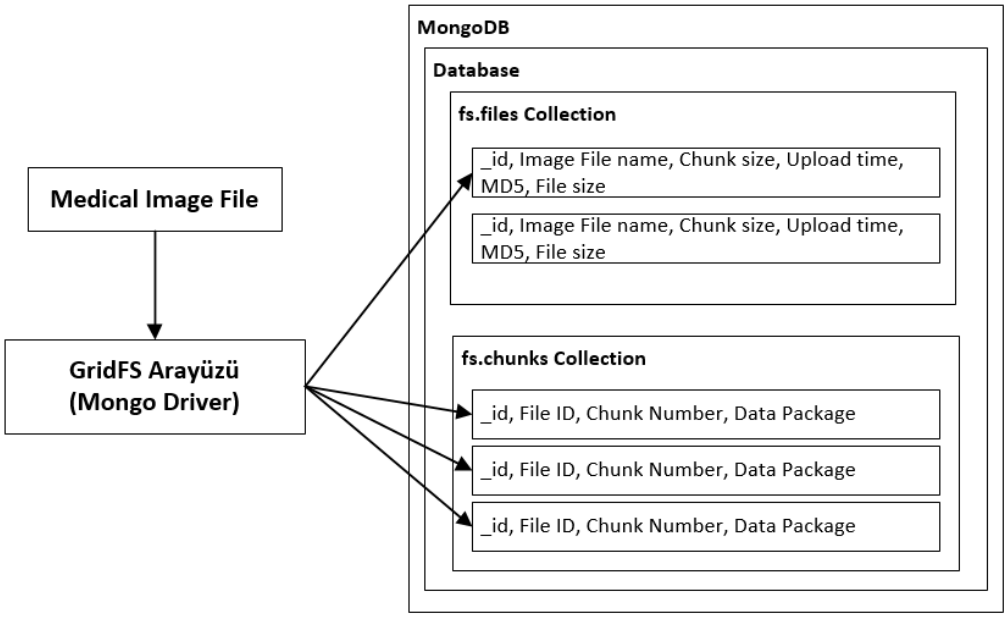


Şekil 3.32. MongoDB bağlantı bekliyor

Veritabanı indeksleri daha hızlı sorgulama için kullanılır. Gelişmiş sistemde, HIMS ID değeri ve vatandaşlık kimlik numarası, hasta adı, soyadı ve tarih verileri bir indeks olarak tanımlanmaktadır.

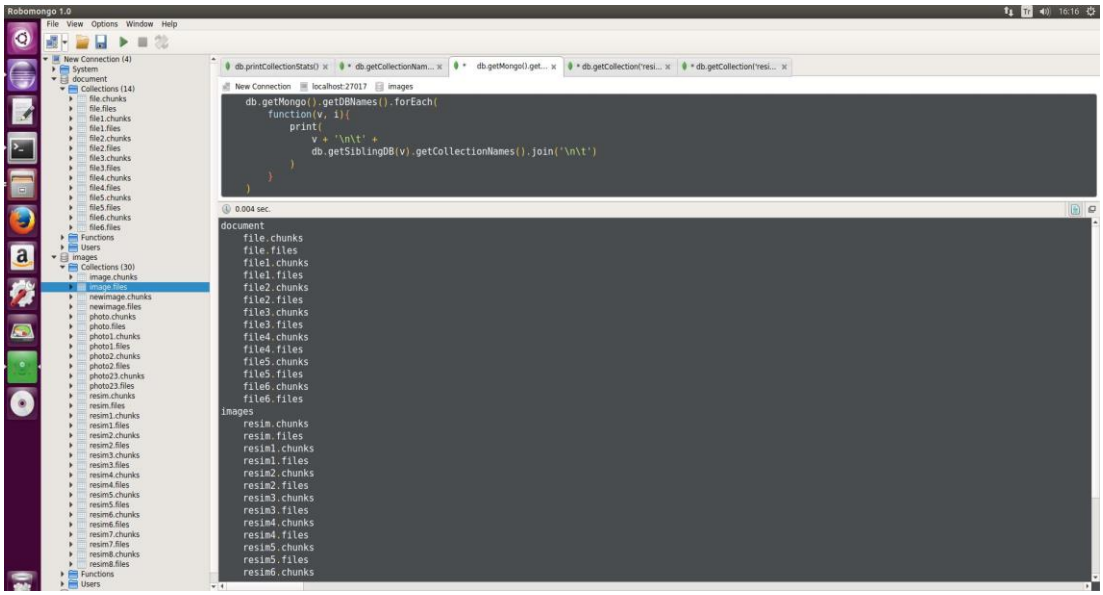
GridFS, büyük dosyaları (varsayılan 16 MB'den büyük) yönetmek için kullanılan bir yerleşik MongoDB dosya sistemidir. GridFS, doğrudan veritabanı sisteminin çoğaltma ve bölümleme mekanizmasını kullanır ve başka bir depolama yapısı gerekmez. GridFS

aracılığıyla veri saklama sonucunda, MongoDB fs.files ve fs.chunks koleksiyonları oluşturur. GridFS mimarisi Şekil 3.33. MongoDB GridFS mimarisinde görülmektedir.

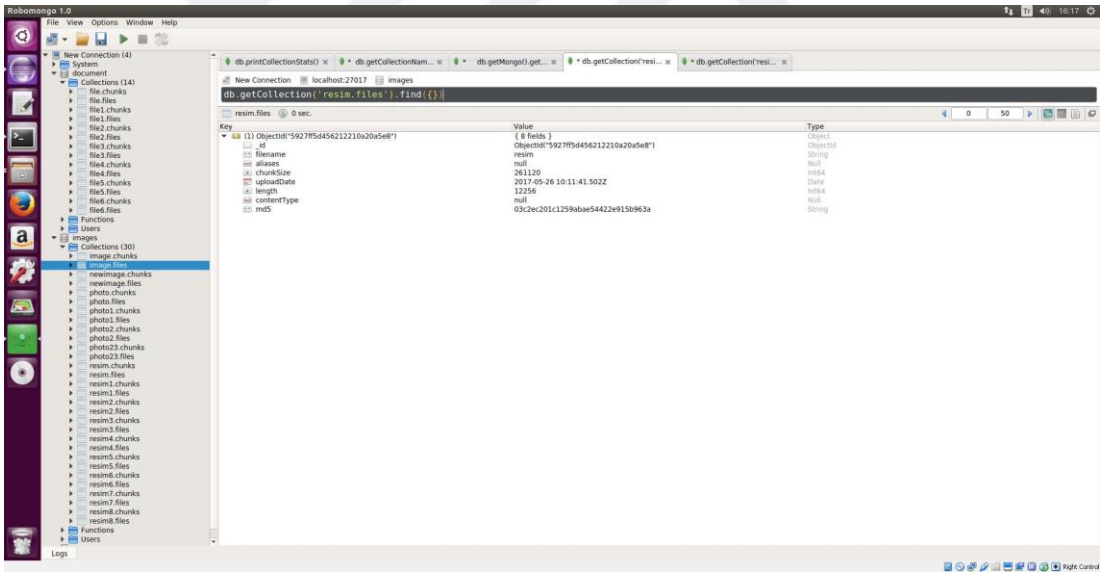


Şekil 3.33. MongoDB GridFS mimarisi

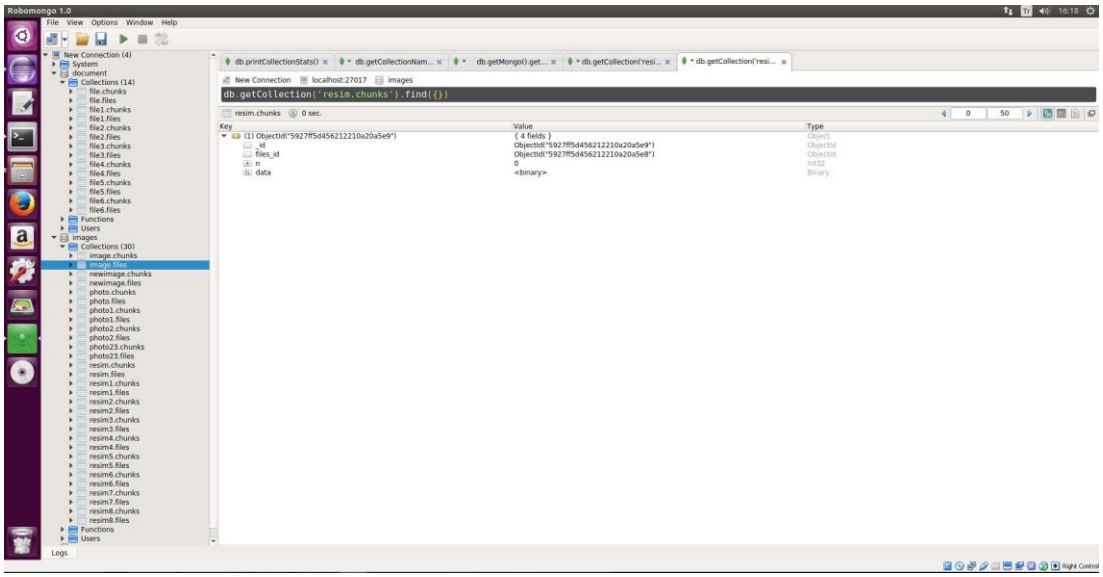
Ayrıca yapılan kurulum sonrasında elde edilen ekran görüntüleri Şekil 3.34. MongoDB tam dizi listesi, Şekil 3.35. MongoDB dosya detay görüntüsü ve Şekil 3.36. MongoDB yığın detay görüntüsü gibidir.



Şekil 3.34. MongoDB tam dizi listesi



Şekil 3.35. MongoDB dosya detay görüntüsü



Şekil 3.36. MongoDB yığın detay görüntüsü

3.4.5. Arama Motoru

Arşivlenen dosyaların tekrar kullanımı için kullanıcı dostu bir arama motoru geliştirilmiştir. MongoDB'de bir dizin olarak tanımlanan geliştirilmiş arama motorunda kullanılan arama kriterleri sayesinde, daha hızlı ve verimli bir arama sunulmaktadır. Arama motorunda bulunan giriş sayfası Şekil 3.37. Arama motoru giriş sayfasında ve arama sayfası Şekil 3.38. Arama motoru arama ve sonuç sayfasında görülmektedir.

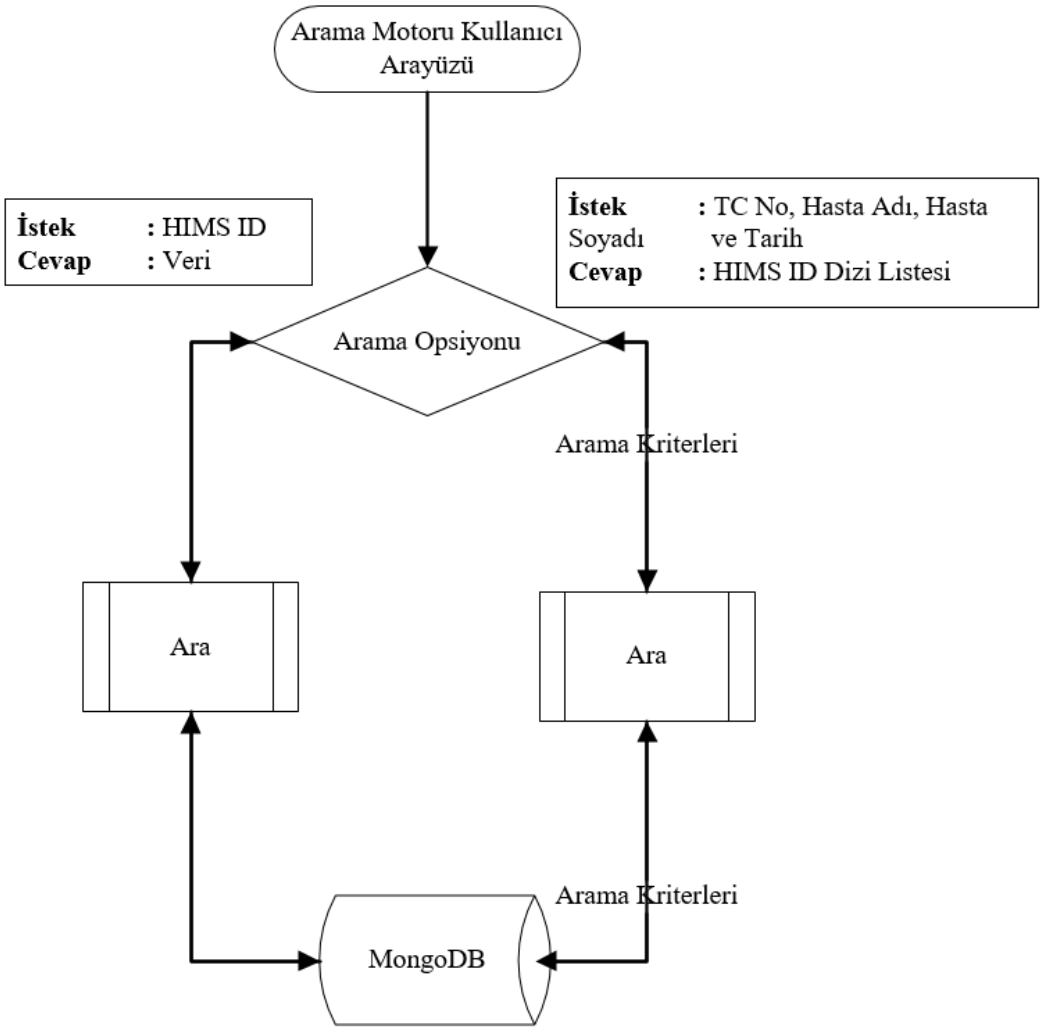


Şekil 3.37. Arama motoru giriş sayfası



Şekil 3.38. Arama motoru arama ve sonuç sayfası

Ayrıca arama motoru kendi içerisinde Şekil 3.39. Arama motoru iş akışında görüldüğü gibi bir akışa sahiptir.



Şekil 3.39. Arama motoru iş akışı

Arama motoru iki farklı yoldan arama yapmanızı sağlar. Birincisi HIMS ID'dir, MongoDB'nin birincil endeksi olarak verimli ve doğrudan arama sağlar. İkinci arama seçeneği vatandaşlık kimlik numarası, hasta adı, hasta soyadı ve tarihtir. Ancak, HIMS kimliği toplama bu yöntemde kullanıcıya listelenir. Kullanıcı istediği seçeneği seçer.

Kullanıcı seçimi yapıldıktan sonra ilk arama yöntemi kullanılır ve veriler HIMS Kimliği ile sorgulanır. Yukarıda belirtildiği gibi, ilk arama yöntemi veri paketini döndürür.

3.4.6. Tersine Önişleme

Arama motoruyla yapılan sorgulamanın sonucu, MongoDB'den alınan veri paketi işlenir ve orijinal medikal görüntü elde edilir. Tıbbi imajın ROI dışı bölgesi aşağıdaki adımlarla oluşturulmuştur: (1) tanımlanan boyut (genişlik, yükseklik) siyah arka plan resmi oluşturulur, (2) elde edilen ve Huffman kodlama algoritması tarafından sıkıştırılan OCR verisinin sonucu veri koordinatıyla değiştirilir. (3) Elde edilen non-ROI bölgesi ve kayıpsız sıkıştırma algoritması tarafından sıkıştırılmış ROI bölgesi birleştirilerek orijinal tıbbi görüntü elde edilir ve kullanıcıya sunulur. Tüm bu adımlar, ön işleme aşamasında olduğu gibi MapReduce program modeliyle Hadoop kümesinde yürütülür.

4. BULGULAR

4.1. Sıkıştırma Sonuçları

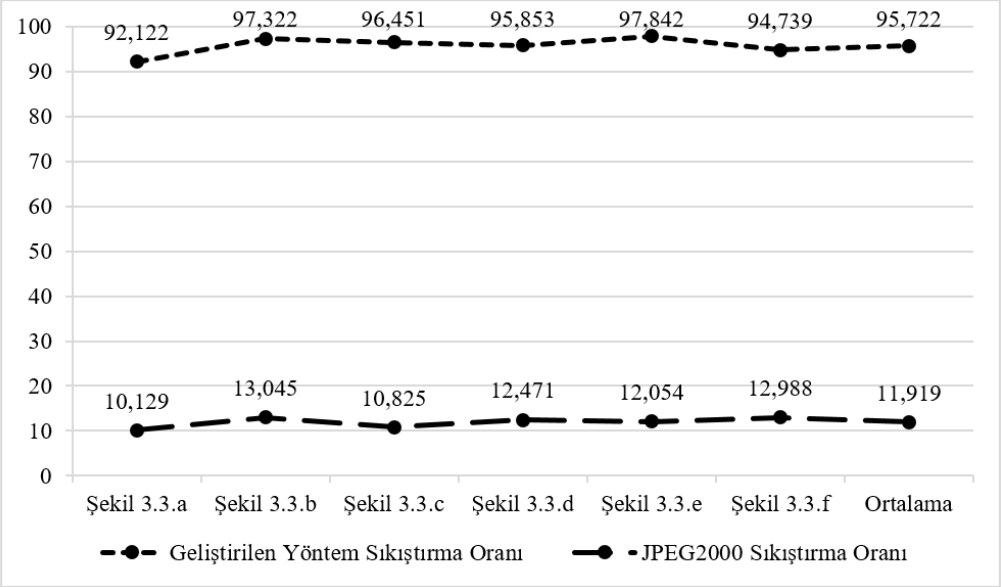
Çalışmada test görüntüleri sıkıştırılmıştır, sıkıştırma sonuçları Çizelge 4.1. Karşılaştırılmalı sıkıştırma oranları gibidir.

Çizelge 4.1. Karşılaştırılmalı sıkıştırma oranları

Görüntü	Görüntü bölümü	Uygulanan algoritma	Önerilen yöntem sıkıştırma oranı	JPEG-2000 sıkıştırma oranı (Non-ROI)
Şekil 3.3. a	ROI	JPEG-LS	3,006	Yok
	Non-ROI	OCR and Huffman	92,122	10,129
Şekil 3.3. b	ROI	JPEG-LS	2,255	Yok
	Non-ROI	OCR and Huffman	97,322	13,045
Şekil 3.3. c	ROI	JPEG-LS	3,916	Yok
	Non-ROI	OCR and Huffman	96,451	10,825
Şekil 3.3. d	ROI	JPEG-LS	3,542	Yok
	Non-ROI	OCR and Huffman	95,853	12,471
Şekil 3.3. e	ROI	JPEG-LS	2,851	Yok
	Non-ROI	OCR and Huffman	97,842	12,054
Şekil 3.3. f	ROI	JPEG-LS	3,246	Yok
	Non-ROI	OCR and Huffman	94,739	12,988

ROI bölümünün sıkıştırma oranı, literatürdeki en son teknoloji yaklaşımına benzerdir [23]. Bu sonuç, kayıpsız sıkıştırma yöntemi nedeniyle normal bir durumdur. Bununla birlikte, görüntünün ROI olmayan bölümü daha iyi sıkıştırma oranına sahiptir ve bu durum çalışmanın başarısına büyük katkı sağlamaktadır. Üçüncüsü, kayıplı sıkıştırma

(JPEG 2000) ve görüntü restorasyon teknikleri görüntünün ROI dışı kısmına uygulanır ve sıkıştırma oranı da yukarıda gösterilmiştir [61, 70]. JPEG2000, algoritmanın görüntünün ROI bölümüne uygulanmamasının nedeni ise bu algoritmanın kayıplı bir sıkıştırma algoritması olmasıdır. Sıkıştırma oranı sonuç şeması Şekil 4.1. Sıkıştırma oranları grafiğinde gösterilmektedir.



Şekil 4.1. Sıkıştırma oranları

Ayrıca, ayrıntılı dosya boyutu bilgileri Çizelge 4.2. Dosya boyutlarında listelenmiştir.

Çizelge 4.2. Dosya boyutları

Görüntü	Orijinal görüntü dosya boyutu (MB)			Sıkıştırılmış görüntü dosya boyutu (MB)		
	ROI	Non-ROI	Toplam	ROI	Non-ROI	Toplam
Şekil 3.3.a	2,560	6,152	8,712	0,852	0,067	0,919
Şekil 3.3.b	3,991	5,240	9,231	1,770	0,054	1,823
Şekil 3.3.c	2,777	5,870	8,647	0,709	0,061	0,770
Şekil 3.3.d	2,318	5,741	8,059	0,654	0,060	0,714
Şekil 3.3.e	2,318	6,164	8,482	0,813	0,063	0,876
Şekil 3.3.f	3,390	5,971	9,361	1,044	0,063	1,107
Average	2,892	5,856	8,749	0,974	0,061	1,035

Çizelge incelendiğinde, geliştirilen sıkıştırma yönteminin ne kadar güçlü olduğu görülebilir. Orijinal görüntülerin ROI bölgesi için ortalama dosya boyutu 2,892 MB'dir ve sıkıştırmadan sonra 0,974 MB'dir. Elde edilen sıkıştırma oranı non-ROI bölgesine göre kayıpsız sıkıştırma uygulandığından ROI bölgesinde daha düşüktür. Bununla birlikte, ROI dışı bölgeden elde edilen veriler, gelişmiş yöntemin başarısı için kritik öneme sahiptir. Orijinal görüntülerin Non-ROI bölgesi ortalama 5,856 MB'dir, ancak sıkıştırmanın uygulanmasının ardından bu değer 0.061 MB'a düşürülmüştür ve bu da bu çalışmanın bir diğer anahtar noktasıdır. Algoritmanın başarısını daha iyi anlamak için, tıbbi görüntü sayısı dikkate alınmalıdır.

Yapılan çalışmada kullanılan sıkıştırma oranı ifadesi orijinal dosya boyutunun sıkıştırılmış dosya boyutuna bölünmesiyle elde edilmiştir.

Ayrıca geliştirilen yöntemin Non-ROI aşamasında OCR sonuçlarına Huffman kodlama algoritması uygulanmaktadır. Algoritma kullanılarak bu bölgenin sıkıştırma oranı artırılmıştır. Huffman kodlama algoritması ile elde edilen test görüntüsü (d) medikal görüntüsünün sıkıştırma sonuçları Çizelge 4.3. Huffman sıkıştırma sonuçlarında gösterilmektedir.

Çizelge 4.3. Huffman sıkıştırma sonuçları

Orijinal dosya boyutu	Sıkıştırılmış dosya boyutu
1580 Byte	812 Byte

Yukarıdaki tabloya göre, metin tabanlı verilerde %48,60 sıkıştırma sonucu elde edilmiştir. Huffman kodlama algoritmasının doğası nedeniyle tekrar eden veri setlerinde daha başarılı sonuçlar elde edilmektedir. Bununla birlikte, tek bir tıbbi imajda bile, üretilen sonuçlar başarılı olmuştur.

Dolayısıyla, önerilen yöntem birçok bakımdan daha verimli, uygun maliyetli ve kullanışlı bir model sunmaktadır.

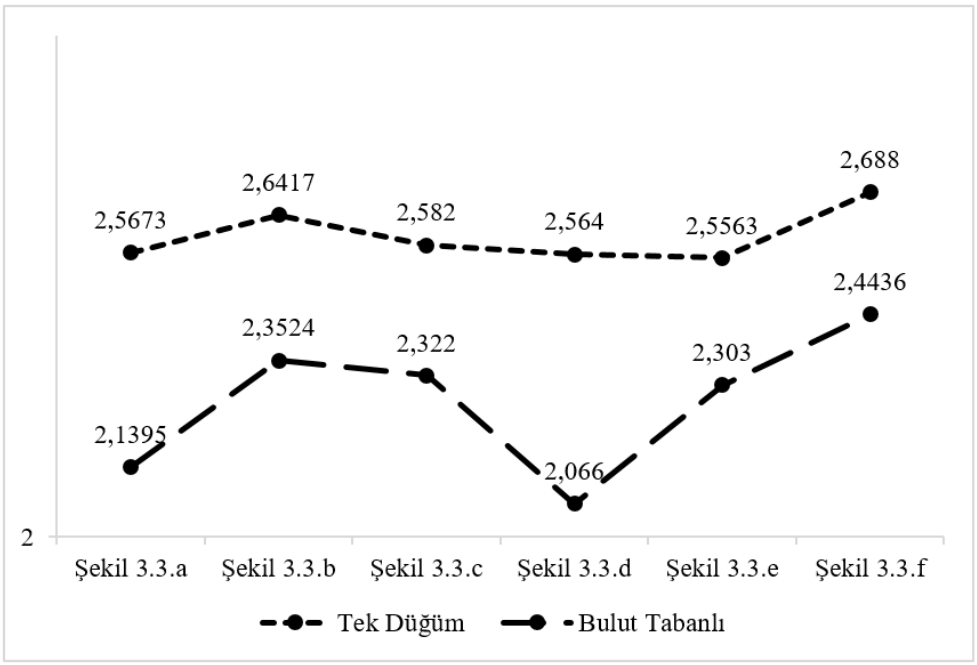
4.2. Bulut Tabanlı Sıkıştırma Sonuçları

Bulut tabanlı geliştirilen medikal görüntü sıkıştırma yaklaşımda sıkıştırma oranları ilk çalışmada olduğu gibidir. Bu çalışma geliştirilen sistemin daha uygulanabilir ve performanslı halidir.

4.2.1. MapReduce/Hadoop Performansı

Yapılan ilk çalışmada görüntü sıkıştırma algoritmaları tek düğüm üzerinden çalıştırılmıştır. Ancak bu sistem bulut tabanlı hale getirilerek geliştirilmiştir. Bu başlıkta önerilen sistemin tek düğüm ve MapReduce/Hadoop tabanlı geliştirilen hali karşılaştırılmaktadır.

Medikal görüntü sıkıştırma paketinin tek düğüm ve MapReduce/Hadoop düğümlerinde işlem hızları Şekil 4.2. İşlem hızlarında görüldüğü gibidir.



Şekil 4.2. İşlem hızları

Mavi renkte gösterilen tek düğüm sonucu olup, turuncu renkte görünen grafik ise bulut tabanlı sonuçlardır.

Yukarıda gösterilen veriler çok yakın değerlere sahiptir, ancak Çizelge 4.4. Toplam işlem süresi karşılaştırma sonuçlarında gösterilen önerilen yöntemin başarısını değerlendirmek için toplam değerleri karşılaştırmak daha doğru olacaktır.

Çizelge 4.4. Toplam işlem süresi karşılaştırma

	Tek düğüm (saniye)	Bulut tabanlı (saniye)
Görüntü Ortalama	15,5993	13,6269
Toplam Süre		8,1537

Tek düğümlü sistem test görüntülerinin işlemlerini 15.5993 saniyede tamamlarken önerilen yöntem 8,1537 saniyede tamamlamıştır. Medikal görüntü paketinde, kümedeki M resim ve N sunucusu olduğu düşünüldüğünde: i) bir düğüm bu işlemleri birer birer gerçekleştirir, ii) önerilen yöntem ortalama M/N işlemleri yapar. Bu bağlamda, önerilen yöntem literatürdeki boşluğu doldurmuştur.

4.2.2. MongoDB Performansı

Geliştirilen bulut tabanlı sistemin tek düğümlü sistemden bir diğer farkı ise NoSQL veri tabanı kullanmasıdır. Önerilen sistem hız, güvenlik ve güvenilirlik için NoSQL veritabanı olan MongoDB'yi kullanmaktadır. Veritabanı hızını ölçerken Çizelge 4.5. Veri tabanı karşılaştırma test sorguları kullanılmıştır.

Çizelge 4.5. Veri tabanı karşılaştırma test sorguları

Sorgu numarası	Sorgu açıklaması
1	Veri yazmak (Veri paketi ve indeksler)
2	Bir sayısal değeri içeren sonuçları almak için ("HIMSID = 52721")
3	Arama motoru ölçütlerini içeren sonuçları almak için (“PatientName = Erdal”, “PatientSurname= Erdal”, “StartDate= 01.01.2016”, “EndData= 01.01.2017”)

Sorguların doğruluğunu artırmak için, her iki veri tabanına da sahte veriler eklenmiştir. Veri tabanlarının yazma hızları Çizelge 4.6. Veri tabanı veri yazma kıyaslamasında sunulmaktadır. Elde edilen sonuçlara göre, önerilen yöntemde kullanılan dağıtık MongoDB, tüm kayıt numaralarında ilişkisel veritabanı olan SQL Server'dan daha hızlıdır.

Çizelge 4.6. Veri tabanı veri yazma kıyaslaması

Sahte kayıt sayısı	Sorgu 1 yanıt süresi (MS)	
	SQL server	Dağıtık-MongoDB
1000	1,37	1,28
10,000	14,98	8,11
100,000	143,08	76,28
1,000,000	1409,32	778,94

Çizelge 4.7. Veri tabanı indeksli sayısal veri çekme kıyaslamasında indekslenmiş sayısal veriyi sorgulayarak elde edilmiştir. Sonuçlara göre SQL Server az miktarda veri ile iyi performans göstermektedir ancak kayıt sayısı arttıkça MongoDB daha iyi performans gösterdiği görülmektedir.

Çizelge 4.7. Veri tabanı indeksli sayısal veri çekme kıyaslaması

Sahte kayıt sayısı	Sorgu 2 yanıt süresi (MS)	
	SQL server	Dağıtık-MongoDB
1000	1,48	4,83
10,000	6,63	3,56
100,000	33,69	25,72
1,000,000	318,50	276,97

Çizelge 4.8. Veri tabanı indeksli olmayan veri çekme kıyaslamasında tabloda indekslenmemiş alanları olan sorgulamanın sonuçları gösterilmektedir. Sonuçlara göre, MongoDB büyük miktarda veri iyi performans göstermektedir.

Çizelge 4.8. Veri tabanı indeksli olmayan veri çekme kıyaslaması

Sahte kayıt sayısı	Sorgu 3 yanıt süresi (MS)	
	SQL server	Dağıtık-MongoDB
1000	3,23	7,56
10,000	13,66	13,95
100,000	79,97	92,44
1,000,000	814,42	650,97

Tabloda görüldüğü gibi, önerilen yöntemde kullanılan MongoDB verisinin sayısı ne kadar yüksek olursa, performans o kadar yüksek olmaktadır. Bu da medikal görüntüler gibi büyük sayılarda veri bulunan uygulamalarda MongoDB kullanımın uygun olacağını göstermektedir.

5. SONUÇLAR

Bu çalışmanın ilk aşamasında, kayıpsız ilgi alanı ve OCR kullanılarak yeni bir medikal görüntü sıkıştırma yöntemi geliştirilmiştir. İlk olarak, orijinal görüntüyü ROI ve ROI olmayan bölümlere ayırmak için bölgeye dayalı aktif kontur yöntemi tüm görüntüye uygulanmıştır. Ardından, kayıpsız sıkıştırma algoritması olan JPEG-LS, görüntünün ROI bölümüne uygulanmaktadır. ROI işlemleri tamamlandığında, OCR işlemi görüntünün ROI olmayan kısmı için uygulanır. OCR yönteminin uygulanması, meta verilerin çıkarılmasını sağlar. OCR işleminden sonra görüntünün ROI dışı kısmı gereğinden fazla ve yok sayılır. Son olarak, daha iyi sıkıştırma oranı sağlamak için çıkarılan verilere Huffman kodlama algoritması uygulanmaktadır. Tüm veriler, depolama alanı ve bant genişliği gereksinimlerini azaltmak için tasarlanmış bir dinamik dosya yapısında saklanır ve arşivlenir.

Geliştirilen sıkıştırma yönteminin başlıca avantajları şunlardır:
(a) ROI ve OCR ile iyi tanımlanmış bir medikal görüntü saklama ve arşivleme tekniğinin uygulanması birleştirilir,
(b) Uygulanan algoritmalar ve tanımlanmış dinamik dosya yapısı sayesinde yüksek sıkıştırma oranına sahip daha etkili ve verimli bir yöntem geliştirilmiştir.

Ayrıca, orijinal test görüntülerinin ortalama dosya boyutu 8,749 MB'dir ve önerilen yöntemle sıkıştırılan test görüntülerinin ortalama dosya boyutu 1,035 MB'dir.

Sonunda ROI bölümünün sıkıştırma oranı, beklendiği gibi literatüre yakın doğru sonuçlar verir. ROI bölgesindeki sonuçlar arasındaki bu benzerlik, kayıpsız sıkıştırma algoritmalarından kaynaklanmaktadır. Ayrıca, önerilen yöntemde, elde edilen sıkıştırma oranı sonuçları, literatürdeki en gelişmiş yaklaşımdan %92,12 ve %97,84 arasında daha iyi sonuçlar verdiği görülmektedir. Bir başka deyişle, literatürdeki önerilen yöntem ve son teknoloji yaklaşımı arasındaki ortalama fark %83,80 olarak tespit edilmiştir. Tüm avantajlarının yanında geliştirilen bu sistem tek düğüm üzerinde çalıştığından düşük performans göstermekte ve uygulanabilirliği azalmaktadır.

Tüm bu nedenler sonucunda çalışmanın ikinci aşamasında, geliştirilen sıkıştırma yöntemi son teknolojiler kullanılarak bulut tabanlı olarak geliştirilmiştir.

Medikal görüntü işleme paketi Hadoop/MapReduce kullanılarak bir sıkıştırma, gerçekleştirildi ve elde edilen veriler dağıtık-MongoDB'de saklandı.

Bu çalışmanın ana katkıları şunlardır:

a) Medikal görüntüler, literatürdeki en son yaklaşımla ve yöntemlerle sıkıştırılmıştır.
b) Tüm işlemler MapReduce programlama modelini kullanarak Hadoop kümesinde gerçekleştirildi.
c) Tüm veriler güvenli bir dağıtık-MongoDB veri tabanında saklanmıştır.
d) Gelişmiş hızlı ve etkin arama motoru, son kullanıcının aramasını yaptığı medikal görüntüye güvenle erişimi sağlamıştır.

Geliştirilen bulut tabanlı çözüm sayesinde daha efektif, verimli ve performanslı bir platform geliştirilmiştir.

Yapılan bu çalışma yalnızca medikal görüntülerin sıkıştırılmasını ve arşivlenmesini kapsamaktadır. Ancak bilgi çağı düşünüldüğünde büyük veri kavramının önemi her geçen gün artmaktadır. Elde edilen bu ham verinin işlenerek bilgiye dönüştürülmesi bir zorunluluktur. Bu nedenle dağıtık mimarilerin gelecekte gelişerek büyük veriler üzerinde daha sık kullanılacakları öngörülmektedir.

Ayrıca, bu çalışma sonucu geliştirilen yöntem ve sonuçlar Science Citation Index Expanded kapsamında bulunan bir dergide yayınlanmak üzere kabul edilmiştir.

Bu çalışma, önerilen yöntemle, yaklaşımla ve elde edilen sonuçlarla gelecekte bu alanda yapılacak olan çalışmalara ışık tutacaktır.

KAYNAKLAR

[1] W. C. G. PEH, "History of the Discovery of X-Rays," *Singapore MED J.*, vol. 36, pp. 437- 441, 1995.

[2] V. Wong and S. Y. Tan, "Medicine in Stamps Harvey Washington Wiley (1844-1930): champion of the Pure Food and Drugs Act," (in English), *Singapore Medical Journal*, vol. 50, no. 3, pp. 235-236, Mar 2009.

[3] H. Wang, F. Guo, D. Feng, and J. Jin, "A signature for content-based image retrieval using a geometrical transform,," presented at the Proceedings of the sixth ACM international conference on Multimedia, Bristol, United Kingdom, September 13 - 16, 1998.

[4] S. T. C. Wong, Ed. *Medical Image Databases*, 1 ed. (The Springer International Series in Engineering and Computer Science, no. 465). p. 398.

[5] D. A. Lisle, *Imaging for Students Fourth Edition*, 4th Edition ed. CRC Press, 2012, p. 312.

[6] (13.04.2017). *Tomography*. Available: https://en.wikipedia.org/wiki/Tomography

[7] P. C. Lauterbur, "Image-Formation by Induced Local Interactions - Examples Employing Nuclear Magnetic-Resonance," (in English), *Clinical Orthopaedics and Related Research*, no. 244, pp. 3-6, Jul 1989.

[8] P. Mansfield, "Multi-planar image formation using NMR spin echoes," *Journal of Physics C: Solid State Physics*, vol. 10, no. 3, 1977.

[9] R. H. Choplin, J. M. Boehme, and C. D. Maynard, "Picture Archiving and Communication-Systems - an Overview," (in English), *Radiographics*, vol. 12, no. 1, pp. 127-129, Jan 1992.

[10] N. H. Strickland, "PACS (picture archiving and communication systems): filmless radiology," (in English), *Archives of Disease in Childhood*, vol. 83, no. 1, pp. 82-86, Jul 2000.

[11] I. D. Robertson and T. Saveraid, "Hospital, radiology, and Picture Archiving and Communication Systems," (in English), *Veterinary Radiology & Ultrasound*, vol. 49, no. 1, pp. S19-S28, Jan-Feb 2008.

[12] K. J. Dreyer, A. Mehta, and J. H. Thrall, "PACS—A Guide to the Digital Revolution," *European Journal of Radiology*, vol. 50, pp. 309–31, 2004.

[13] Nema. (2004, 21.04). *Digital Imaging and Communications in Medicine (DICOM)* Available: http://dicom.nema.org/standard.html

[14] H. Oosterwijk, T. Edition, Ed. *DICOM Basics* O Tech, 2005.

[15] J. J. McConnell, *Computer Graphics Companion*, 1 ed. Wiley, p. 388

[16] P. Mildenberger, M. Eichelberg, and E. Martin, "Introduction to the DICOM standard," (in English), *European Radiology,* vol. 12, no. 4, pp. 920-927, Apr 2002.

[17] A. C. Bovik, *Handbook of Image and Video Processing*, 1st edition ed. (Communications, Networking and Multimedia). Academic Press, 2000, p. 891.

[18] E. Tuncel, *Klinik Radyoloji*. Nobel & Güneş Tıp Kitabevi Ltd.Şti., 2008, p. 1052.

[19] M. Nelson, *The Data Compression Book*, 2 Edition ed. Wiley, 1995.

[20] K. Sayood, *Introduction to Data Compression*, 4 ed. (The Morgan Kaufmann Series in Multimedia Information and Systems). Morgan Kaufmann, 2012, p. 768.

[21] N. Özmen, "Image Segmentation And Smoothing Via Partial Differentıal Equations," Yüksek Lisans, Orta Doğu Teknik Üniversitesi, 2009.

[22] M. Kass, A. Witkin, and D. Terzopoulos, "Snakes - Active Contour Models," (in English), *International Journal of Computer Vision,* vol. 1, no. 4, pp. 321-331, 1987.

[23] Z. Y. Zuo, X. Lan, L. H. Deng, S. K. Yao, and X. P. Wang, "An improved medical image compression technique with lossless region of interest," (in English), *Optik,* vol. 126, no. 21, pp. 2825-2831, 2015.

[24] L. Eikvil, *Optical Character Recognition*. Oslo: Norsk Regnesentral, 1993.

[25] B. M. Sagar, G. Shobha, and P. K. Ramakanth, "OCR for printed Kannada text to Machine editable format using Database approach," (in English), *Proceedings of the 9th Wseas International Conference on Automation and Information,* pp. 322-+, 2008.

[26] F. Sheng, A. Bilgin, P. J. Sementilli, and M. W. Marcellin, "Lossy and lossless image compression using reversible integer wavelet transforms," (in English), *1998 International Conference on Image Processing - Proceedings, Vol 3,* pp. 876-880, 1998.

[27] B. Xiao, G. Lu, Y. H. Zhang, W. S. Li, and G. Y. Wang, "Lossless image compression based on integer Discrete Tchebichef Transform," (in English), *Neurocomputing*, vol. 214, pp. 587-593, Nov 19 2016.

[28] D. A. Huffman, "A Method for the Construction of Minimum-Redundancy Codes," in *Proceedings of the IRE*, 2007 pp. 1098 - 1101.

[29] R. P. Loce and E. R. Dougherty, "Facilitation of optimal binary morphological filter design via structuring element libraries and design constraints," *Opt. Eng.*, vol. 31, pp. 1008-1025 1992.

[30] H. P. Kramer and J. B. Bruckner, "Iterations of a nonlinear transformation for enhancement of digital images.," *Pattern Recog.*, vol. 7, pp. 53-58, 1975.

[31] S. J. Lee, R. M. Haralick, and L. G. Shapiro, "Morphologic edge detection," *IEEE Trans. Rob. Autom.*, pp. 142-156, 1987.

[32] H. J. A. M. Heijmans, *Morphological Image Operators*. Academic Press, 1994, p. 509.

[33] S. Ghemawat, H. Gobioff, and S.-T. Leung, "The Google File System," presented at the 19th ACM Symposium on Operating Systems Principles, Lake George, NY, October 2003, 2003.

[34] J. Dean and S. Ghemawat, "MapReduce: Simplified Data Processing on Large Clusters " presented at the OSDI'04: Sixth Symposium on Operating System Design and Implementation, San Francisco, CA, December, 2004, 2004.

[35] E. Baldeschwieler. (2008, 25.04). *Yahoo! Launches World's Largest Hadoop Production Application*. Available: http://yahoohadoop.tumblr.com/post/98098649696/yahoo-launches-worlds-largest-hadoop-production

[36] R. C. Taylor, "An overview of the Hadoop/MapReduce/HBase framework and its current applications in bioinformatics,," in *Proceedings of the 11th Annual Bioinformatics Open Source Conference (BOSC)*, 2010, vol. 11.

[37] T. White, *Hadoop: The Definitive Guide*, 4th Edition ed. O'Reilly Media, 2015, p. 728.

[38] C. J. Driscoll, D. Lyon, and N. L. McCain, "Integrating Genomics Into Biobehavioral Research: A Transplantation Exemplar," (in English), *Biological Research for Nursing*, vol. 13, no. 4, pp. 340-345, Oct 2011.

[39] J. Ekanayake, T. Gunarathne, and J. Qiu, "Cloud Technologies for Bioinformatics Applications," (in English), *Ieee Transactions on Parallel and Distributed Systems,* vol. 22, no. 6, pp. 998-1011, Jun 2011.

[40] P. Andlinger. (2013, 12). *RDBMS dominate the database market, but NoSQL systems are catching up.* Available: http://db-engines.com/en/blog_post/23

[41] K. Banker, *MongoDB in Action,* 1st Edition ed. Manning Publications, p. 312.

[42] K. Chodorow, *MongoDB: The Definitive Guide*, 2nd Edition ed. O'Reilly Media, 2013, p. 409.

[43] Db-Engines. (15.04.2017). *Homepage*.

[44] J. Yoon, D. Jeong, C. H. Kang, and S. Lee, "Forensic investigation framework for the document store NoSQL DBMS: MongoDB as a case study," (in English), *Digital Investigation,* vol. 17, pp. 53-65, Jun 2016.

[45] J. Strom and P. C. Cosman, "Medical image compression with lossless regions of interest," (in English), *Signal Processing,* vol. 59, no. 2, pp. 155-171, Jun 1997.

[46] S. B. Gokturk, C. Tomasi, B. Girod, and C. Beaulieu, "Medical image compression based on region of interest, with application to colon CT images," (in English), *Proceedings of the 23rd Annual International Conference of the Ieee Engineering in Medicine and Biology Society, Vols 1-4*, vol. 23, pp. 2453-2456, 2001.

[47] M. Kaur and V. Wasson, "ROI Based Medical Image Compression for Telemedicine Application," (in English), *Proceedings of the 4th International Conference on Eco-Friendly Computing and Communication Systems,* vol. 70, pp. 579-585, 2015.

[48] U. Iqbal *et al.*, "Cancer-disease associations: A visualization and animation through medical big data," (in English), *Computer Methods and Programs in Biomedicine,* vol. 127, pp. 44-51, Apr 2016.

[49] E. AbuKhousa, N. Mohamed, and J. Al-Jaroodi, "e-Health Cloud: Opportunities and Challenges," *Future Internet,* vol. 4, no. 3, pp. 621-645, 2012.

[50] G. C. Kagadis *et al.*, "Cloud computing in medical imaging," (in English), *Medical Physics,* vol. 40, no. 7, Jul 2013.

[51] V. Chang and G. Wills, "A model to compare cloud and non-cloud storage of Big Data," (in English), *Future Generation Computer Systems-the International Journal of Escience,* vol. 57, pp. 56-76, Apr 2016.

[52] W. M. Lin, W. C. Dou, Z. J. Zhou, and C. Liu, "A cloud-based framework for Home-diagnosis service over big medical data," (in English), *Journal of Systems and Software,* vol. 102, pp. 192-206, Apr 2015.

[53] S. Istephan and M. R. Siadat, "Unstructured medical image query using big data - An epilepsy case study," (in English), *Journal of Biomedical Informatics,* vol. 59, pp. 218-226, Feb 2016.

[54] Q. Yao, Y. Tian, P. F. Li, L. L. Tian, Y. M. Qian, and J. S. Li, "Design and Development of a Medical Big Data Processing System Based on Hadoop," (in English), *Journal of Medical Systems,* vol. 39, no. 3, Mar 2015.

[55] Z. Goli-Malekabadi, M. Sargolzaei-Javan, and M. K. Akbari, "An effective model for store and retrieve big health data in cloud computing," (in English), *Computer Methods and Programs in Biomedicine,* vol. 132, pp. 75-82, Aug 2016.

[56] W. Xin, "Design and Implementation of CNEOST Image Database Based on NoSQL System," *Chinese Astronomy and Astrophysics,* vol. 38, no. 2, pp. 211-221, 2014.

[57] Y. T. Liao *et al.*, "Data adapter for querying and transformation between SQL and NoSQL database," (in English), *Future Generation Computer Systems-the International Journal of Escience,* vol. 65, pp. 111-121, Dec 2016.

[58] Y. S. Li and S. Manoharan, "A performance comparison of SQL and NoSQL databases," (in English), *2013 Ieee Pacific Rim Conference on Communications, Computers and Signal Processing (Pacrim),* pp. 15-19, 2013.

[59] P. Getreuer, "Chan–Vese Segmentation," *Image Processing On Line,* vol. 2, pp. 214–224, 2012.

[60] D. Kaur and Y. Kaur, "Various Image Segmentation Techniques:A Review," *International Journal of Computer Science and Mobile Computing,* vol. 3, no. 5, pp. 809-814, 2014.

[61] K. Saini, M. L. Dewal, and M. Rohit, "A Fast Region-Based Active Contour Model for Boundary Detection of Echocardiographic Images," (in English), *Journal of Digital Imaging,* vol. 25, no. 2, pp. 271-278, Apr 2012.

[62] Y. Wu. (2009, 28.05.2017). *Chan Vese Active Contours without edges*. Available: http://www.mathworks.com/matlabcentral/fileexchange/23445-chan-vese-active-contours-without-edges?focused=5111672&tab=function

[63] S. Lankton. (2016, 28.05.2017). *Active Contour Segmentation*. Available: https://www.mathworks.com/matlabcentral/fileexchange/19567-active-contour-segmentation?focused=6125387&tab=function

[64] M. s. Team. (2015, 28.05.2017). *DICOM to any Image Format Converter*. Available: https://www.mathworks.com/matlabcentral/fileexchange/49882-dicom-to-any-image-format-converter?focused=3863747&tab=function

[65] MathWorks. (28.05.2017). *Multithresh*. Available: https://www.mathworks.com/help/images/ref/multithresh.html

[66] M. J. Weinberger, G. Seroussi, and G. Sapiro, "The LOCO-I lossless image compression algorithm: Principles and standardization into JPEG-LS," (in English), *Ieee Transactions on Image Processing,* vol. 9, no. 8, pp. 1309-1324, Aug 2000.

[67] M. Grabner. (2015, 28.05.2017). *Basic JPEG-LS (LOCO-I) Encoder*. Available: https://www.mathworks.com/matlabcentral/fileexchange/46424-jpegls-encode-zip

[68] (28.05.2017). *OCR*. Available: https://www.mathworks.com/help/vision/ref/ocr.html

[69] (28.05.2017). *Huffmanenco*. Available: https://www.mathworks.com/help/comm/ref/huffmanenco.html

[70] Y. F. Lou, X. Q. Zhang, S. Osher, and A. Bertozzi, "Image Recovery via Nonlocal Operators," (in English), *Journal of Scientific Computing,* vol. 42, no. 2, pp. 185-197, Feb 2010.

EKLER

EK 1

Chan-Vese aktif çevre yöntemi kodları

Çalıştırma Dosyası

```
close all
clear all

I = imread('GörüntüDosyası');

m = zeros(size(I,1),size(I,2));
m(20:120,20:120) = 1;
seg = chenvese(I,m,4000,0.1,'chan'); % ability on gray image
```

Fonksiyon Dosyası

```
function seg = chenvese(I,mask,num_iter,mu,method)

 if(~exist('mu','var'))
  mu=0.2;
 end

 if(~exist('method','var'))
  method = 'chan';
 end

  s = 200./min(size(I,1),size(I,2));
  if s<1
     I = imresize(I,s);
  end
 if ischar(mask)
    switch lower (mask)
       case 'small'
          mask = maskcircle2(I,'small');
       case 'medium'
          mask = maskcircle2(I,'medium');
       case 'large'
          mask = maskcircle2(I,'large');
       case 'whole'
          mask = maskcircle2(I,'whole');
       case 'whole+small'
          m1 = maskcircle2(I,'whole');
          m2 = maskcircle2(I,'small');
          mask = zeros(size(I,1),size(I,2),2);
          mask(:,:,1) = m1(:,:,1);
          mask(:,:,2) = m2(:,:,2);
       otherwise
          error('unrecognized mask shape name (MASK).');
    end
 else
     if s<1
        mask = imresize(mask,s);
     end
    if size(mask,1)>size(I,1) || size(mask,2)>size(I,2)
       error('dimensions of mask unmathch those of the image.')
    end
```

```
    switch lower(method)
       case 'multiphase'
          if  (size(mask,3) == 1)
             error('multiphase requires two masks but only gets one.')
          end
    end
 end

 switch lower(method)
  case 'chan'
     if size(I,3)== 3
        P = rgb2gray(uint8(I));
        P = double(P);
     elseif size(I,3) == 2
        P = 0.5.*(double(I(:,:,1))+double(I(:,:,2)));
     else
        P = double(I);
     end
     layer = 1;

  case 'vector'
     s = 200./min(size(I,1),size(I,2));
     I = imresize(I,s);
     mask = imresize(mask,s);
     layer = size(I,3);
     if layer == 1
        display('only one image component for vector image')
     end
     P = double(I);

  case 'multiphase'
     layer = size(I,3);
     if size(I,1)*size(I,2)>200^2
        s = 200./min(size(I,1),size(I,2));
        I = imresize(I,s);
        mask = imresize(mask,s);
     end

     P = double(I);
  otherwise
     error('!invalid method')
end
switch lower(method)
  case {'chan','vector'}
     mask = mask(:,:,1);
     phi0 = bwdist(mask)-bwdist(1-mask)+im2double(mask)-.5;

     force = eps;
      figure();
      subplot(2,2,1); imshow(I); title('Input Image');
      subplot(2,2,2); contour(flipud(phi0), [0 0], 'r','LineWidth',1); title('initial contour');
      subplot(2,2,3); title('Segmentation');

      for n=1:num_iter
         inidx = find(phi0>=0);
         outidx = find(phi0<0);
         force_image = 0;
         for i=1:layer
            L = im2double(P(:,:,i));
```

```
            c1 = sum(sum(L.*Heaviside(phi0)))/(length(inidx)+eps);
            c2 = sum(sum(L.*(1-Heaviside(phi0))))/(length(outidx)+eps);
            force_image=-(L-c1).^2+(L-c2).^2+force_image;
          end
          force = mu*kappa(phi0)./max(max(abs(kappa(phi0))))+1/layer.*force_image;

          force = force./(max(max(abs(force))));

          dt=0.5;

          old = phi0;
          phi0 = phi0+dt.*force;
          new = phi0;
          indicator = checkstop(old,new,dt);

          if(mod(n,20) == 0)
            showphi(I,phi0,n);
          end;
          if indicator
            showphi(I,phi0,n);

            seg = phi0<=0;

            subplot(2,2,4); imshow(seg); title('Region - Of - Interest');
            imsave(seg);
            return;
          end
        end;
        showphi(I,phi0,n);

        seg = phi0<=0;

        subplot(2,2,4); imshow(seg); title('Region - Of - Interest');
    case 'multiphase'
        mask1 = mask(:,:,1);
        mask2 = mask(:,:,2);
        phi1=bwdist(mask1)-bwdist(1-mask1)+im2double(mask1)-.5;
        phi2=bwdist(mask2)-bwdist(1-mask2)+im2double(mask2)-.5;
        figure();
        subplot(2,2,1);
        if layer ~= 1
          imshow(I); title('Input Image');
        else
          imagesc(P); axis image; colormap(gray);title('Input Image');
        end
        subplot(2,2,2);
        hold on
        contour(flipud(mask1),[0,0],'r','LineWidth',2.5);
        contour(flipud(mask1),[0,0],'x','LineWidth',1);
        contour(flipud(mask2),[0,0],'g','LineWidth',2.5);
        contour(flipud(mask2),[0,0],'x','LineWidth',1);
        title('initial contour');
        hold off
        subplot(2,2,3); title('Segmentation');

        for n=1:num_iter
          nb1 = find(phi1<1.2 & phi1>=-1.2);
          inidx1 = find(phi1>=0);
          outidx1 = find(phi1<0);
```

```
nb2 = find(phi2<1.2 & phi2>=-1.2);
inidx2 = find(phi2>=0);
cc11 = intersect(inidx1,inidx2);
cc12 = intersect(inidx1,outidx2);
cc21 = intersect(outidx1,inidx2);
cc22 = intersect(outidx1,outidx2);

f_image11 = 0;
f_image12 = 0;
f_image21 = 0;
f_image22 = 0;
for i=1:layer
   L = im2double(P(:,:,i));

if isempty(cc11)
   c11 = eps;
else
   c11 = mean(L(cc11));
end

if isempty(cc12)
   c12 = eps;
else
   c12 = mean(L(cc12));
end

if isempty(cc21)
   c21 = eps;
else
   c21 = mean(L(cc21));
end

if isempty(cc22)
   c22 = eps;
else
   c22 = mean(L(cc22));
end



f_image11=(L-c11).^2.*Heaviside(phi1).*Heaviside(phi2)+f_image11;
f_image12=(L-c12).^2.*Heaviside(phi1).*(1-Heaviside(phi2))+f_image12;
f_image21=(L-c21).^2.*(1-Heaviside(phi1)).*Heaviside(phi2)+f_image21;
f_image22=(L-c22).^2.*(1-Heaviside(phi1)).*(1-Heaviside(phi2))+f_image22;
end

curvature1 = mu*kappa(phi1);
curvature1 = curvature1(nb1);

fim1 = 1/layer.*(-f_image11(nb1)+f_image21(nb1)-f_image12(nb1)+f_image22(nb1));
fim1 = fim1./max(abs(fim1)+eps);


curvature2 = mu*kappa(phi2);
curvature2 = curvature2(nb2);

fim2 = 1/layer.*(-f_image11(nb2)+f_image12(nb2)-f_image21(nb2)+f_image22(nb2));
fim2 = fim2./max(abs(fim2)+eps);
```

```
    force1 = curvature1+fim1;
    force2 = curvature2+fim2;

    dt = 1.5;

    old(:,:,1) = phi1;
    old(:,:,2) = phi2;

    phi1(nb1) = phi1(nb1)+dt.*force1;
    phi2(nb2) = phi2(nb2)+dt.*force2;

    new(:,:,1) = phi1;
    new(:,:,2) = phi2;


    indicator = checkstop(old,new,dt);

    if indicator
      showphi(I, new, n);
      seg11 = (phi1>0 & phi2>0);
      seg12 = (phi1>0 & phi2<0);
      seg21 = (phi1<0 & phi2>0);
      seg22 = (phi1<0 & phi2<0);

      se = strel('disk',1);
      aa1 = imerode(seg11,se);
      aa2 = imerode(seg12,se);
      aa3 = imerode(seg21,se);
      aa4 = imerode(seg22,se);
      seg = aa1+2*aa2+3*aa3+4*aa4;

      subplot(2,2,4); imagesc(seg);axis image;title('Global Region-Based Segmentation');


       return
    end
    phi1 = reinitialization(phi1, 0.6);
    phi2 = reinitialization(phi2, 0.6);

    if(mod(n,20) == 0)
      phi(:,:,1) = phi1;
      phi(:,:,2) = phi2;
      showphi(I, phi, n);
    end;
  end;
  phi(:,:,1) = phi1;
  phi(:,:,2) = phi2;
  showphi(I, phi, n);
seg11 = (phi1>0 & phi2>0);
seg12 = (phi1>0 & phi2<0);
seg21 = (phi1<0 & phi2>0);
seg22 = (phi1<0 & phi2<0);


se = strel('disk',1);
aa1 = imerode(seg11,se);
aa2 = imerode(seg12,se);
aa3 = imerode(seg21,se);
aa4 = imerode(seg22,se);
```

```
    seg = aa1+2*aa2+3*aa3+4*aa4;
    subplot(2,2,4); imagesc(seg);axis image;title('Region - Of - Interest');
end
```

EK 2

Bölge tabanlı aktif kontur yöntemi kodu

Çalıştırma Dosyası

```
I = imread('GörüntüDosyası');
m = zeros(size(I,1),size(I,2));
m(111:222,123:234) = 1;

I = imresize(I,.5);
m = imresize(m,.5);

subplot(2,2,1); imshow(I); title('Input Image');
subplot(2,2,2); imshow(m); title('Initialization');
subplot(2,2,3); title('Segmentation');

seg = region_seg(I, m, 4000);

subplot(2,2,4); imshow(seg); title('Region-Based Segmentation');
```

Fonksiyon Dosyası

```
function seg = region_seg(I,init_mask,max_its,alpha,display)

 if(~exist('alpha','var'))
  alpha = .2;
 end
 if(~exist('display','var'))
  display = true;
 end
 I = im2graydouble(I);

 phi = mask2phi(init_mask);

 for its = 1:max_its
  idx = find(phi <= 1.2 & phi >= -1.2);
  upts = find(phi<=0);
  vpts = find(phi>0);
  u = sum(I(upts))/(length(upts)+eps);
  v = sum(I(vpts))/(length(vpts)+eps);

  F = (I(idx)-u).^2-(I(idx)-v).^2;
  curvature = get_curvature(phi,idx);

  dphidt = F./max(abs(F)) + alpha*curvature;

  dt = .45/(max(dphidt)+eps);

  phi(idx) = phi(idx) + dt.*dphidt;

  phi = sussman(phi, .5);


  if((display>0)&&(mod(its,20) == 0))
   showCurveAndPhi(I,phi,its);
  end
 end

 if(display)
  showCurveAndPhi(I,phi,its);
```

```
  end

  seg = phi<=0;

function showCurveAndPhi(I, phi, i)
  imshow(I,'initialmagnification',200,'displayrange',[0 255]); hold on;
  contour(phi, [0 0], 'g','LineWidth',4);
  contour(phi, [0 0], 'k','LineWidth',2);
  hold off; title([num2str(i) ' Iterations']); drawnow;

function phi = mask2phi(init_a)
  phi=bwdist(init_a)-bwdist(1-init_a)+im2double(init_a)-.5;

function curvature = get_curvature(phi,idx)
    [dimy, dimx] = size(phi);
    [y x] = ind2sub([dimy,dimx],idx);

    ym1 = y-1; xm1 = x-1; yp1 = y+1; xp1 = x+1;

    ym1(ym1<1) = 1; xm1(xm1<1) = 1;
    yp1(yp1>dimy)=dimy; xp1(xp1>dimx) = dimx;


    idup = sub2ind(size(phi),yp1,x);
    iddn = sub2ind(size(phi),ym1,x);
    idlt = sub2ind(size(phi),y,xm1);
    idrt = sub2ind(size(phi),y,xp1);
    idul = sub2ind(size(phi),yp1,xm1);
    idur = sub2ind(size(phi),yp1,xp1);
    iddl = sub2ind(size(phi),ym1,xm1);
    iddr = sub2ind(size(phi),ym1,xp1);


    phi_x  = -phi(idlt)+phi(idrt);
    phi_y  = -phi(iddn)+phi(idup);
    phi_xx = phi(idlt)-2*phi(idx)+phi(idrt);
    phi_yy = phi(iddn)-2*phi(idx)+phi(idup);
    phi_xy = -0.25*phi(iddl)-0.25*phi(idur)...
            +0.25*phi(iddr)+0.25*phi(idul);
    phi_x2 = phi_x.^2;
    phi_y2 = phi_y.^2;


    curvature = ((phi_x2.*phi_yy + phi_y2.*phi_xx - 2*phi_x.*phi_y.*phi_xy)./...
             (phi_x2 + phi_y2 +eps).^(3/2)).*(phi_x2 + phi_y2).^(1/2);

function img = im2graydouble(img)
  [dimy, dimx, c] = size(img);
  if(isfloat(img))
    if(c==3)
      img = rgb2gray(uint8(img));
    end
  else
    if(c==3)
      img = rgb2gray(img);
    end
    img = double(img);
  end
```

```
function D = sussman(D, dt)
  a = D - shiftR(D);
  b = shiftL(D) - D;
  c = D - shiftD(D);
  d = shiftU(D) - D;

  a_p = a;  a_n = a;
  b_p = b;  b_n = b;
  c_p = c;  c_n = c;
  d_p = d;  d_n = d;

  a_p(a < 0) = 0;
  a_n(a > 0) = 0;
  b_p(b < 0) = 0;
  b_n(b > 0) = 0;
  c_p(c < 0) = 0;
  c_n(c > 0) = 0;
  d_p(d < 0) = 0;
  d_n(d > 0) = 0;

  dD = zeros(size(D));
  D_neg_ind = find(D < 0);
  D_pos_ind = find(D > 0);
  dD(D_pos_ind) = sqrt(max(a_p(D_pos_ind).^2, b_n(D_pos_ind).^2) ...
                  + max(c_p(D_pos_ind).^2, d_n(D_pos_ind).^2)) - 1;
  dD(D_neg_ind) = sqrt(max(a_n(D_neg_ind).^2, b_p(D_neg_ind).^2) ...
                  + max(c_n(D_neg_ind).^2, d_p(D_neg_ind).^2)) - 1;

  D = D - dt .* sussman_sign(D) .* dD;

function shift = shiftD(M)
  shift = shiftR(M')';

function shift = shiftL(M)
  shift = [ M(:,2:size(M,2)) M(:,size(M,2)) ];

function shift = shiftR(M)
  shift = [ M(:,1) M(:,1:size(M,2)-1) ];

function shift = shiftU(M)
  shift = shiftL(M')';

function S = sussman_sign(D)
  S = D ./ sqrt(D.^2 + 1);
```

EK 3

Otsu eşikleme yöntemi kodu

Çalıştırma ve fonksiyon dosyaları

```
%%
close all;
clear all;
clc;
%%

%%DICOM to PNG
[X,MAP]=dicomread('GörüntüDosyası');
image8 = uint8(255 * mat2gray(X));
%imwrite(image8,'temp.png', 'png');% Save Image as png format

%%
Im_gr=image8;
%Im_gr=imread('temp.png');
%Im_gr= rgb2gray(Im_org);

%% Otsu Segmentation Algortimas?
[thresh,metric] = multithresh(Im_gr,1);
seg_I = imquantize(Im_gr,thresh);
RGBLabeled = label2rgb(seg_I);
%%

%Transparent backround yapıyoruz (roi için)
[X,Y] = size(Im_gr);
B = zeros([X,Y]);%Background alpha matrisi
R = zeros([X,Y]);%ROI alpha matrisi

for xi=1:X
  for xj=1:Y
    if(seg_I(xi,xj) ~= 1)   %% Assuming uint8, 255 would be white
      R(xi,xj) = 1;      %% Assign 1 to transparent color(white)
    else
      B(xi,xj)=1;
    end
  end
end

imwrite(Im_gr,'roi.png','Alpha',R);
imwrite(Im_gr,'bck.png','Alpha',B);
```

EK 4

JPEG-LS kodu

Çalıştırma ve Fonksiyon Dosyası

```
function [Compress Recover] = JPEGLS_codec(image, useRunMode) %

global CREGIONS; global CONTEXTS1; global CONTEXTS; global EOR_CONTEXTS; global
TOT_CONTEXTS; global EOR_0;
global EOLINE; global NOEOLINE; global MAX_C; global MIN_C; global MELCSTATES;
global vLUT; global classmap; global zeroLUT;
global T1; global T2; global T3; global J; global C; global B; global N; global A;
global BITBUFSIZE; global bits; global Reg; global fp; global buff; global lutmax; global bpp;
global qbpp;
global Alpha; global ceil_half_Alpha; global Reset; global limit; global Cols; global limit_reduce;
global near; global melcstate; global melclen; global melcorder;
global decbuff; global decfp;

SIZE = size(image);
height = SIZE(1);
width = SIZE(2);

ImageSize = height*width; Cols = width;
image = double(image);

CREGIONS = 9; CONTEXTS1 = CREGIONS*CREGIONS*CREGIONS; CONTEXTS =
(CONTEXTS1+1)/2;
EOR_CONTEXTS = 2; TOT_CONTEXTS = CONTEXTS + EOR_CONTEXTS; EOR_0 =
CONTEXTS+1;
EOLINE = 1; NOEOLINE = 0; MAX_C = 127; MIN_C = -128; MELCSTATES = 32;
Alpha0 = 255;
Alpha = Alpha0+1; ceil_half_Alpha =  ceil(Alpha/2);
Reset = 64; lutmax = 256;
bpp = max(2, ceil(log2(Alpha))); qbpp = bpp;
vLUT = zeros(3, 2 * lutmax); classmap = zeros(1, CONTEXTS1); zeroLUT = zeros(1, 256);
T1 = 3; T2 = 7; T3 = 21;
limit = 4*bpp - qbpp - 1;
C = zeros(1,TOT_CONTEXTS);
B = zeros(1,TOT_CONTEXTS);
N = ones(1,TOT_CONTEXTS);
A_init = max(2,floor((Alpha+2^5)/2^6));
A = ones(1,TOT_CONTEXTS) * A_init;
J = [0,0,0,0,1,1,1,1,2,2,2,2,3,3,3,3,4,4,5,5,6,6,7,7,8,9,10,11,12,13,14,15];
pscanline = zeros(1, width+1);
BITBUFSIZE = 32; bits = BITBUFSIZE; Reg = uint32(0); fp = 1; buff = zeros(1, height*width);
melcstate = 1; melclen = J(melcstate); melcorder = bitsll(1, melclen);
prepareLUTs();

psl_0 = 0;
for row=1:height
   ['Encoding row : ' num2str(row) ]
   cscanline = image(row,:);
   sl_0 = pscanline(1);
   psl=pscanline;
   sl=cscanline;
   lossless_doscanline(psl, sl, psl_0, sl_0, useRunMode);
   pscanline = [cscanline cscanline(end)];
   psl_0 = sl_0;
end
```

```
while (bits < 32)
   temp = bitsrl(Reg, 24);
   buff(1, fp) = temp;
   fp = fp + 1;
   if ( temp == 255 )
      bits = bits + 7;
      Reg = bitsll(Reg, 7);
      Reg = bitand(Reg, bitcmp(uint32(bitsll(1, BITBUFSIZE-1))));
   else
      bits = bits + 8;
      Reg = bitsll(Reg, 8);
   end
end
fp = fp - 1;
Compress = buff;
['Oubput is ' num2str(fp*8) ' bits.']
decbuff = [buff 0 0 0 0 0 0 0 0 0 0 0 ];
C = zeros(1,TOT_CONTEXTS); B = zeros(1,TOT_CONTEXTS); N = ones(1,TOT_CONTEXTS);
A = ones(1,TOT_CONTEXTS) * A_init;
melcstate = 1; melclen = J(melcstate); melcorder = bitsll(1, melclen);
decfp = 1; bits = 0; Reg = uint32(0);
fillbuffer(24);
createzeroLUT();
pscanline = zeros(1, width+1);  psl_0 = 0;
Recover = zeros(height, width);
for row=1:height
   sl_0 = pscanline(1);
   psl=pscanline;
   cscanline = lossless_undoscanline(psl, width, psl_0, sl_0, useRunMode);
   Recover(row, :) = cscanline;
   imerr = cscanline - image(row, :);
   if length(find(imerr ~= 0)) > 0
      ['Decoding row : ' num2str(row) ', Error occurs in this row' ]
   else
      ['Decoding row : ' num2str(row)]
   end
   pscanline = [cscanline cscanline(end)];
   psl_0 = sl_0;
end
Recover = uint8(Recover);
imshow(Recover);
end

function cl = clip(x, Range)
cl=x;
if(x < 0)
   cl = 0;
elseif (x >= Range)
   cl = Range - 1;
end
end

function Px= predict(Rb, Ra, Rc)
if (Rb > Ra)
   minx = Ra;
   maxx = Rb;
else
   maxx = Ra;
   minx = Rb;
```

```
end
if (Rc >= maxx)
   Px = minx;
elseif (Rc <= minx)
   Px = maxx;
else
   Px = Ra + Rb - Rc;
end
end

function putbits(x, n)
global BITBUFSIZE; global bits; global Reg; global fp; global buff;
bits = bits - n;
Reg = bitor(Reg, bitsll(uint32(x), bits));
while (bits <= 24)
   temp = bitsrl(Reg, 24);
   buff(1, fp) = temp;
   fp = fp + 1;
   if ( temp == 255 )
      bits = bits + 7;
      Reg = bitsll(Reg, 7);
      Reg = bitand(Reg, bitcmp(uint32(bitsll(1, BITBUFSIZE-1))));
   else
      bits = bits + 8;
      Reg =  bitsll(Reg, 8);
   end
end
end

function put_ones(hits)
if ( hits < 24 )
   putbits(bitsll(1, hits)-1,hits);
else
   nn = hits;
   while ( nn >= 24 )
      putbits(bitsll(1, 24)-1,24);
      nn = nn - 24;
   end
   if ( nn > 0 )
      putbits(bitsll(1, nn)-1, nn);
   end
end
end

function put_zeros(n)
global bits; global Reg; global fp; global buff;
bits = bits - n;
while (bits <= 24)
   temp = bitsrl(Reg, 24);
   buff(1, fp) = temp ;
   fp = fp + 1;
   Reg = bitsll(Reg, 8);
   bits = bits + 8;
end
end

function process_run(runlen, eoline)
global MELCSTATES; global EOLINE; global J; global melcstate; global melclen; global
melcorder; global limit_reduce;
```

```
hits = 0;
while ( runlen >= melcorder )
   hits = hits + 1;
   runlen = runlen - melcorder;
   if ( melcstate < MELCSTATES )
      melcstate = melcstate + 1;
      melclen = J(melcstate);
      melcorder = bitsll(1, melclen);
   end
end
put_ones(hits);
if ( eoline==EOLINE )
   if ( runlen )
      put_ones(1);
   end
else
   limit_reduce = melclen+1;
   putbits(runlen,limit_reduce);
   if ( melcstate > 1 )
      melcstate = melcstate - 1;
      melclen = J(melcstate);
      melcorder = bitsll(1, melclen);
   end
end
end

function lossless_end_of_run(Ra, Rb, lx, RItype)
global EOR_0; global B; global N; global A; global Alpha; global ceil_half_Alpha; global Reset;
global limit; global limit_reduce; global qbpp;
Q = EOR_0 + RItype;
Nt = N(Q);
At = A(Q);
Errval = lx - Rb;
if (RItype)
   At = At + floor(Nt/2);
elseif ( Rb < Ra )
   Errval = -Errval;
end
for k=0:32
   if Nt >= At
      break;
   end
   Nt = bitsll(Nt, 1);
end
if (Errval < 0)
   Errval = Errval + Alpha;
end
if( Errval >= ceil_half_Alpha )
   Errval = Errval - Alpha;
end
oldmap = double( k==0 && Errval && bitsll(B(Q), 1) < Nt );
if( Errval < 0)
   MErrval = int16(-bitsll(Errval, 1)-1-RItype+oldmap);
   B(Q) = B(Q) + 1;
else
   MErrval = int16(bitsll(Errval, 1)-RItype-oldmap);
end
absErrval = bitsrl(int16(MErrval+1-RItype), 1);
A(Q) = A(Q) + absErrval;
```

```
if (N(Q) == Reset)
   N(Q) = floor(N(Q)/2);
   A(Q) = floor(A(Q)/2);
   B(Q) = floor(B(Q)/2);
end
N(Q) = N(Q)+1;
eor_limit = limit - limit_reduce;
unary = bitsrl(int16(MErrval), k);
if ( unary < eor_limit )
   put_zeros(unary);
   putbits(bitsll(1,k) + bitand(uint32(MErrval), uint32(bitsll(1,k)) - 1), k + 1);
else
   put_zeros(eor_limit);
   putbits(bitsll(1,qbpp) + MErrval-1, qbpp+1);
end
end

function lossless_regular_mode(Q, SIGN, Px, lx)
global C; global B; global N; global A; global MAX_C; global MIN_C; global Alpha; global
ceil_half_Alpha; global limit; global Reset; global qbpp;
Nt = N(Q);
At = A(Q);
Px = Px + SIGN*C(Q);
Px = clip(Px, Alpha);
Errval = SIGN * (lx - Px);
nst = Nt;
for k=0:32
   if nst >= At
      break;
   end
   nst = bitsll(nst, 1);
end
Bt = B(Q);  temp = double( k==0 && (bitsll(Bt, 1) <= -Nt));
if (Errval < 0)
   Errval = Errval + Alpha;
end
if (Errval >= ceil_half_Alpha)
   Errval = Errval - Alpha;
   absErrval = -Errval;
   MErrval = int16(bitsll(absErrval, 1) - 1 - temp);
else
   absErrval = Errval;
   MErrval = int16(bitsll(absErrval, 1) + temp);
end
Bt = Bt + Errval;
B(Q) = Bt;
A(Q) = A(Q) + absErrval;
if (Nt == Reset)
   Nt = floor(Nt/2);
   N(Q) = Nt;
   A(Q) = floor(A(Q)/2);
   Bt = floor(Bt/2);
   B(Q) = Bt;
end
Nt = Nt+1;
N(Q) = Nt;
if ( Bt <= -Nt )
   if (C(Q) > MIN_C)
      C(Q) = C(Q) - 1;
```

```
    end
    B(Q) = B(Q) + Nt;
    if ( B(Q) <= -Nt )
        B(Q) = -Nt+1;
    end
elseif ( Bt > 0 )
    if (C(Q) < MAX_C)
        C(Q) = C(Q) + 1;
    end
    B(Q) = B(Q) - Nt;
    if ( B(Q) > 0 )
        B(Q) = 0;
    end
end
unary = bitsrl(MErrval, k);
if ( unary < limit )
    put_zeros(unary);
    putbits(bitsll(1, k) + bitand(MErrval, bitsll(1, k) - 1), k + 1);
else
    put_zeros(limit);
    putbits(bitsll(1, qbpp) + MErrval - 1, qbpp+1);
end
end

function lossless_doscanline(psl, sl, psl_0, sl_0, useRunMode)
global EOLINE; global NOEOLINE; global vLUT; global lutmax; global classmap; global Cols;
i=1;
Rc = psl_0;
Rb = psl(1);
Ra = sl_0;
eol = NOEOLINE;
while (i <= Cols)
    lx = sl(i);
    Rd = psl(i+1);
    cont = vLUT(1, Rd-Rb+lutmax+1) + vLUT(2, Rb-Rc+lutmax+1) + vLUT(3, Rc-Ra+lutmax+1);
    if(cont == 0 && useRunMode)
        RUNcnt = 0;
        if lx == Ra
            while (1)
                RUNcnt = RUNcnt + 1;
                i = i+1;
                if i > Cols
                    process_run(RUNcnt, EOLINE);
                    eol = EOLINE;
                    break;
                end
                lx = sl(i);
                if lx ~= Ra
                    Rd = psl(i+1);
                    Rb = psl(i);
                    break;
                end
            end
        end
        if eol ~= EOLINE
            process_run(RUNcnt,NOEOLINE);
            RItype = 0;
            if (Ra==Rb)
                RItype = 1;
```

```
            end
            lossless_end_of_run(Ra, Rb, lx, RItype);
        end
    else
Px = predict(Rb, Ra, Rc);
        cont = classmap(cont+1);
        if (cont<0)
            SIGN = -1;
            cont = -cont;
        else
            SIGN = 1;
        end
        Q = cont + 1;
        lossless_regular_mode(Q, SIGN, Px, lx);
    end
    if eol ~= EOLINE
        Ra = lx;
        Rc = Rb;
        Rb = Rd;
        i = i+1;
    end
end
end

function prepareLUTs()
global T1; global T2; global T3; global vLUT; global classmap; global CREGIONS; global
CONTEXTS1;
idx = 0;
for i = -255:255
    if ( i <= -T3  )
        idx = 7;
    elseif ( i <= -T2 )
        idx = 5;
    elseif ( i <= -T1 )
        idx = 3;
    elseif ( i <= -1 )
        idx = 1;
    elseif ( i == 0 )
        idx = 0;
    elseif ( i < T1 )
        idx = 2;
    elseif ( i < T2 )
        idx = 4;
    elseif ( i < T3 )
        idx = 6;
    else
        idx = 8;
    end
    vLUT(1, i + 256 + 1) = CREGIONS * CREGIONS * idx;
    vLUT(2, i + 256 + 1) = CREGIONS * idx;
    vLUT(3, i + 256 + 1) = idx;
end
j=0;
for i=1:CONTEXTS1-1
    if classmap(i+1) == 0
        q1 = floor(i/(CREGIONS*CREGIONS));
        q2 = mod(floor(i/CREGIONS), CREGIONS);
        q3 = mod(i, CREGIONS);
        sgn = 1;
```

```
      if(mod(q1,2)==1 || (q1==0 && mod(q2,2)==1) || (q1==0 && q2==0 && mod(q3,2)==1))
         sgn = -1;
      else
         sgn = 1;
      end
      n1 = 0; n2 = 0; n3 = 0;
      if (q1)
         if mod(q1, 2) == 1
            n1 = q1 + 1;
         else
            n1 = q1 - 1;
         end
      end
      if (q2)
         if mod(q2, 2) == 1
            n2 = q2 + 1;
         else
            n2 = q2 - 1;
         end
      end
      if (q3)
         if mod(q3, 2) == 1
            n3 = q3 + 1;
         else
            n3 = q3 - 1;
         end
      end
      ineg = (n1*CREGIONS+n2)*CREGIONS+n3;
      j = j+1;
      classmap(i+1) = sgn*j;
      classmap(ineg+1) = -sgn*j;
   end
end
end
function fillbuffer(no)
global Reg; global bits; global decbuff; global decfp;
Reg = bitsll(Reg, no);
bits = bits + no;
while (bits >= 0)
   x = decbuff(decfp);
   decfp = decfp + 1;
   if ( x == 255 )
      if ( bits < 8 )
         decfp = decfp - 1;
         decbuff(decfp)= 255;
         break;
      else
         x = uint32(decbuff(decfp));
         decfp = decfp + 1;
         if bitand(x,128) == 0
            Reg = bitor(Reg, bitor(bitsll(uint32(255), bits), bitsll(bitand(x, 127), bits-7)));
            bits = bits - 15;
         else
            Reg = bitor(Reg, bitor(bitsll(uint32(255), bits), bitsll(x, bits-8)));
            bits = bits - 16;
         end
         continue;
      end
   end
```

```
    Reg = bitor(Reg, bitsll(x, bits));
    bits = bits - 8;
end
end

function createzeroLUT()
global zeroLUT;
j = 1;
k = 1;
l = 8;
for i = 1:256
    zeroLUT(i) = l;
    k = k-1;
    if (k == 0)
        k = j;
        l = l - 1;
        j = bitsll(j, 1);
    end
end
end

function temp = getbits(no)
global Reg; global BITBUFSIZE;
temp = double(bitsrl(Reg, BITBUFSIZE - no));
fillbuffer(no);
end

function current = lossless_regular_mode_d(Q, SIGN, Px)
global Reg; global B; global C; global N; global A; global Alpha; global Reset; global limit; global
zeroLUT; global qbpp; global MAX_C; global MIN_C;
absErrval = 0;
Nt = N(Q);
At = A(Q);
nst = Nt;
for k=0:32
    if nst >= At
        break;
    end
    nst = bitsll(nst, 1);
end
while (1)
    temp = zeroLUT(bitsrl(Reg, 24)+1);
    absErrval = absErrval + temp;
    if (temp ~= 8)
        fillbuffer(temp + 1);
        break;
    end
    fillbuffer(8);
end
if ( absErrval < limit )
    if (k)
        absErrval = bitsll(absErrval, k);
        absErrval = absErrval + getbits(k);
    end
else
    absErrval = getbits(qbpp) + 1;
end

if bitand(int16(absErrval), 1)
```

```
    absErrval = (absErrval + 1) / 2;
    Errval = -absErrval;
else
    absErrval = absErrval/2;
    Errval = absErrval;
end
Bt = B(Q);
if ( k==0 && (2*Bt <= -Nt) )
    Errval = -(Errval+1);
    if Errval<0
        absErrval = -Errval;
    else
        absErrval = Errval;
    end
end
if ( SIGN == -1 )
    Px = Px - C(Q);
    Px = clip(Px, Alpha);
    current = Px - Errval;
else
    Px = Px + C(Q);
    Px = clip(Px, Alpha);
    current = Px + Errval;
end
if (current < 0)
    current = current + Alpha;
elseif (current >= Alpha)
    current = current - Alpha;
end
Bt = Bt + Errval;
B(Q) = Bt;
A(Q) = A(Q) + absErrval;
if(Nt == Reset)
    Nt = floor(Nt/2);
    N(Q) = Nt;
    A(Q) = floor(A(Q)/2);
    Bt = floor(Bt/2);
    B(Q) = Bt;
end
Nt = Nt + 1;
N(Q) = Nt;
if  ( Bt <= -Nt )
    if (C(Q) > MIN_C)
        C(Q) = C(Q) - 1;
    end
    B(Q) = B(Q) + Nt;
    Bt = B(Q);
    if ( Bt <= -Nt )
        B(Q) = -Nt+1;
    end
elseif ( Bt > 0 )
    if (C(Q) < MAX_C)
        C(Q) = C(Q) + 1;
    end
    B(Q) = B(Q) - Nt;
    Bt = B(Q);
    if ( Bt > 0 )
        B(Q) = 0;
    end
```

```
end
end

function LEFT = process_run_dec(lineleft) %
global Reg; global J; global melcstate; global melclen; global melcorder; global zeroLUT; global
EOLINE; global NOEOLINE; global limit_reduce; global MELCSTATES;
runlen = 0;
eol = NOEOLINE;
while eol == NOEOLINE
   temp = zeroLUT(bitcmp(uint8(bitsrl(Reg, 24)))+1);
   for hits = 1:temp
      runlen = runlen + melcorder;
      if ( runlen >= lineleft )
         if (runlen==lineleft && melcstate < MELCSTATES)
            melcstate = melcstate + 1;
            melclen=J(melcstate);
            melcorder=bitsll(1, melclen);
         end
         fillbuffer(hits);
         LEFT = lineleft;
         eol = EOLINE;
         break;
      end
      if eol ~= EOLINE
         if ( melcstate < MELCSTATES )
            melcstate = melcstate + 1;
            melclen = J(melcstate);
            melcorder = bitsll(1, melclen);
         end
      end
   end
   if eol ~= EOLINE
      if (temp ~= 8)
         fillbuffer(temp + 1);
         break;
      end
      fillbuffer(8);
   end
end
if eol ~= EOLINE
   if ( melclen)
      temp = getbits(melclen);
      runlen = runlen + temp;
   end
   limit_reduce = melclen+1;
   if ( melcstate > 1 )
      melcstate = melcstate -1;
      melclen = J(melcstate);
      melcorder = bitsll(1, melclen);
   end
   LEFT = runlen;
end
end

function Ix = lossless_end_of_run_d(Ra, Rb, RItype)
global Reg; global EOR_0; global B; global N; global A; global limit; global limit_reduce; global
Reset; global zeroLUT; global qbpp; global Alpha;
MErrval=0;
Q = EOR_0 + RItype;
```

```
Nt = N(Q);
At = A(Q);
if ( RItype )
   At = At + floor(Nt/2);
end
for k=0:32
   if Nt >= At
      break;
   end
   Nt = Nt * 2;
end
while 1
   temp = zeroLUT(bitsrl(Reg, 24)+1);
   MErrval = MErrval + temp;
   if (temp ~= 8)
      fillbuffer(temp + 1);
      break;
   end
   fillbuffer(8);
end
eor_limit = limit - limit_reduce;
if ( MErrval < eor_limit )
   if (k)
      MErrval = bitsll(MErrval, k);
      MErrval = MErrval + getbits(k);
   end
else
   MErrval = getbits(qbpp) + 1;
end
oldmap = double(( k==0 && (RItype || MErrval) && (2*B(Q)<Nt))); %
MErrval = MErrval + ( RItype + oldmap );
if ( bitand(int16(MErrval), 1) )
   Errval = oldmap - floor((MErrval+1)/2);
   absErrval = -Errval-RItype;
   B(Q) = B(Q) + 1;
else
   Errval = floor(MErrval/2);
   absErrval = Errval - RItype;
end
if ( Rb < Ra )
   Ix = Rb - Errval;
else
   Ix = Rb + Errval;
end
if (Ix < 0)
   Ix = Ix + Alpha;
elseif (Ix >= Alpha)
   Ix = Ix - Alpha;
end
 A(Q) = A(Q) + absErrval;
if (N(Q) == Reset)
   N(Q) = floor(N(Q)/2);
   A(Q) = floor(A(Q)/2);
   B(Q) = floor(B(Q)/2);
end
N(Q) = N(Q) + 1;
end

function sl = lossless_undoscanline(psl, width, psl_0, sl_0, useRunMode) %
```

```
global EOLINE; global NOEOLINE; global vLUT; global classmap; global lutmax;
eol = NOEOLINE;
i=1;
sl = zeros(1, width);
Rc = psl_0;
Rb = psl(1);
Ra = sl_0;
while (i <= width)
   Rd = psl(i + 1);
   cont =  vLUT(1, Rd - Rb + lutmax+1) + vLUT(2, Rb - Rc + lutmax+1) + vLUT(3, Rc - Ra +
lutmax+1);
   if ( cont == 0 && useRunMode)
      n = process_run_dec(width-i+1);
      m = n;
      if ( m > 0 )
         while (n > 0)
            sl(i) = Ra;
            i = i + 1;
            n = n-1;
         end
         if (i > width)
            eol = EOLINE;
            break;
         end
         if eol ~= EOLINE
            Rb = psl(i);
            Rd = psl(i + 1);
         end
      end
      RItype = 0;
      if (Ra==Rb)
         RItype = 1;
      end
      Ra = lossless_end_of_run_d(Ra, Rb, RItype);
   else
      Px = predict(Rb, Ra, Rc);
      cont = classmap(cont+1);
      if (cont < 0)
         SIGN = -1;
         cont = -cont;
      else
         SIGN = +1;
      end
      Q = cont + 1;
      Ra = lossless_regular_mode_d(Q, SIGN, Px);
   end
   sl(i) = Ra;
   Rc = Rb;
   Rb = Rd;
   i = i + 1;
end
end
```

EK 5

OCR kodu

```
clc;
clear all;
close all;

imageName = ‘GörüntüDosyası’;
I = imread(imageName);
% Perform OCR
results = ocr(I);
len = length(results.Words)

for i=1:len
   i
   % Display one of the recognized words
   word = results.Words{i}
   % Location of the word in I
   wordBBox = results.WordBoundingBoxes(i,:)
   Iname = insertObjectAnnotation(I, 'rectangle', wordBBox, word);
   imshow(Iname);
end
```

EK 6

Huffman kodu

Sınıf Kodu

```
classdef Huffman
  properties
    leftNode = []
    rightNode = []
    probability
    code = ''
    character
  end
end
```

Döngü Kodu

```
function f = loop(tempNode,codec)
if ~isempty(tempNode)
codec = [codec tempNode.code];

if ~isempty(tempNode.character)
disp(tempNode.character);
disp(codec);
end
if ~isempty(tempNode.code)

end
loop(tempNode.leftNode,codec);
loop(tempNode.rightNode,codec);
end
f = codec;
end
```

Çalıştırma Kodu

```
clc;
clear all;
close all;

a = [0.1 0.1 0.2 0.2 0.4];
b = ['a' 'b' 'c' 'd' 'e'];

thearray = Huffman.empty(5,0);

for i=1:length(a)
   thearray(i).probability = a(i);
   thearray(i).character = b(i);
end

temparray = thearray;

for k = 1:size(temparray,2)-1
   for i=1:size(temparray,2)
      for j = 1:size(temparray,2)-1
         if (temparray(j).probability > temparray(j+1).probability)

         end
      end
   end
```

```
    newnode = Huffman;
    newnode.probability = temparray(1).probability + temparray(2).probability;
     temparray(1).code = '0';
     temparray(2).code = '1';

    newnode.leftNode = temparray(1);
    newnode.rightNode = temparray(2);

     temparray = temparray(3:size(temparray,2));

     temparray = [newnode temparray];

end

rootNode = temparray(1);
codec = '';

loop(rootNode,codec);

disp(codec)
```

EK 7

Hadoop Optik Karakter Tanıma Kodu

Ana Kod

```
package ocr.mr;

import com.google.common.io.Files;
import net.sourceforge.tess4j.Tesseract;
import net.sourceforge.tess4j.TesseractException;
import org.apache.commons.io.IOUtils;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;
import java.io.ByteArrayInputStream;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.NullWritable;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.conf.Configured;

import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.Mapper;
import org.apache.hadoop.mapreduce.Reducer;
import org.apache.hadoop.mapreduce.Job;
import org.apache.hadoop.mapreduce.lib.input.FileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.FileSplit;
import org.apache.hadoop.mapreduce.lib.output.FileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.SequenceFileOutputFormat;
import org.apache.hadoop.mapreduce.lib.output.TextOutputFormat;
import org.apache.hadoop.util.Tool;
import org.apache.hadoop.util.ToolRunner;

import org.apache.commons.io.IOUtils;
import javax.imageio.ImageIO;
import java.awt.image.BufferedImage;

import java.io.File;
import java.io.IOException;
import java.io.InputStream;
import java.io.InputStreamReader;
import java.net.URL;
import java.net.URI;
import java.util.Arrays;
import java.util.Iterator;

public class OCRMR extends Configured implements Tool {
        public static class Map extends Mapper<Text, BytesWritable, Text, Text> {

                private Text word = new Text();
                private Text filepath;
```

```
		private Text filename;

		Tesseract instance = null;

		@Override
			protected void setup(Context context) throws IOException,
InterruptedException {

				File tessdata = new File("tessdata.tar.gz/tessdata");
				File nativelibs = new File("tesslib.tar.gz");
				try {
					for( File lib : nativelibs.listFiles() ) {
						System.load(lib.getCanonicalPath());
					}

					instance = Tesseract.getInstance();

	instance.setDatapath(tessdata.getCanonicalPath());
					instance.setTessVariable("LC_NUMERIC", "C");
				} catch (IOException e) {
					System.out.println("Failed to obtain Tessdata
library or folder");
				}

			}

		public void map(Text key, BytesWritable data, Context context)
			throws IOException, InterruptedException {

				InputStream is = new
ByteArrayInputStream(data.getBytes());
				BufferedImage image =
ImageIO.read(IOUtils.toBufferedInputStream(is));

				String result = null;
				try {
					result = instance.doOCR(image);
				} catch (TesseractException e) {
					e.printStackTrace();
				}
				if (!result.isEmpty()) {
					word.set(result);
					context.write(key, word);
				}
			}
	}

	public static class Reduce extends Reducer<Text, Text, Text, Text> {

		public void reduce(Text filename, Text ocrText, Context context)
			throws IOException, InterruptedException {
				context.write(filename, new Text(ocrText));
			}
	}
```

```
	//@Override
		public int run(String[] args) throws Exception {
			Job job = Job.getInstance(getConf());
			job.setJarByClass(OCRMR.class);
			job.setJobName("OCR");

			job.setOutputKeyClass(Text.class);
			job.setOutputValueClass(Text.class);

			job.setMapperClass(Map.class);
			job.setCombinerClass(Reduce.class);
			job.setReducerClass(Reduce.class);

			job.setInputFormatClass(BinaryFileInputFormat.class);
			job.setOutputFormatClass(TextOutputFormat.class);

			FileInputFormat.setInputPaths(job, new Path(args[0]));
			FileOutputFormat.setOutputPath(job, new Path(args[1]));

			job.addCacheArchive(new URI("/tmp/tessdata.tar.gz"));
			job.addArchiveToClassPath(new org.apache.hadoop.fs.Path(new
URI("/tmp/tesslib.tar.gz")));

			return job.waitForCompletion(true) ? 0 : 1;
		}

	public static void main(String[] args) throws Exception {
		int exitCode = ToolRunner.run(new Configuration(), new OCRMR(),
args);
		System.exit(exitCode);
	}

}
```

Anahtar İşlemleri

```
package ocr.mr;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.IOUtils;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.CombineFileSplit;

class BinaryFileRecordReader extends RecordReader<Text, BytesWritable> {
	private CombineFileSplit split;
	private FileSystem fs;
```

```
        private Text key;
        private BytesWritable value;
        private Path[] paths;
        private FSDataInputStream currentStream;
        private int count = 0;
        private boolean processed = false;

        public BinaryFileRecordReader(CombineFileSplit split, TaskAttemptContext
context)
                        throws IOException
        {
                this.paths = split.getPaths();
                this.fs = FileSystem.get(context.getConfiguration());
                this.split = split;
        }

        @Override
        public void initialize(InputSplit split, TaskAttemptContext context)
                        throws IOException, InterruptedException {
        }

        @Override
        public boolean nextKeyValue() throws IOException, InterruptedException {
                if (count >= split.getNumPaths())
                {
                        processed = true;
                        return false; // we have no more data to parse
                }

                Path path = null;
                key = new Text();
                value = new BytesWritable();

                try
                {
                        path = this.paths[count];
                } catch (Exception e) {
                        return false;
                }

                currentStream = null;
                try {
                        currentStream = fs.open(path);
                        key.set(path.getName());
                        int length = (int) fs.getFileStatus(path).getLen();
                        byte[] contents = new byte[ length ];
               IOUtils.readFully(currentStream, contents, 0, length);
               value.set(contents, 0, length);
          } finally {
                                currentStream.close();
          }
                count++;
                return true; // we have more data to parse
        }
```

```
	@Override
	public Text getCurrentKey() throws IOException, InterruptedException {
		return key;
	}

	@Override
	public BytesWritable getCurrentValue() throws IOException,
InterruptedException {
		return value;
	}

	@Override
	public float getProgress() throws IOException {
		return processed ? 1.0f : (float) (count / paths.length);
	}

	@Override
	public void close() throws IOException {
		processed = true;
	}
}
```

Kayıt İşlemleri

```
package ocr.mr;

import java.io.IOException;

import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.Text;
import org.apache.hadoop.mapreduce.InputSplit;
import org.apache.hadoop.mapreduce.JobContext;
import org.apache.hadoop.mapreduce.RecordReader;
import org.apache.hadoop.mapreduce.TaskAttemptContext;
import org.apache.hadoop.mapreduce.lib.input.CombineFileInputFormat;
import org.apache.hadoop.mapreduce.lib.input.CombineFileSplit;

public class BinaryFileInputFormat extends
		CombineFileInputFormat<Text, BytesWritable> {
	@Override
	protected boolean isSplitable(JobContext context, Path file) {
		return false;
	}

	@Override
	public RecordReader<Text, BytesWritable> createRecordReader(
			InputSplit split, TaskAttemptContext context) throws
IOException {

		return new BinaryFileRecordReader((CombineFileSplit) split, context);
	}
}
```

EK 8

Hadoop Huffman kodlama algoritması kod bloğu

Ana Kod

```
package br.ufrj.ppgi.huffmanhadoop;

public class Main {

	public static void main(String[] args) throws Exception {
		// TODO Auto-generated method stub
		String in = new String(Defines.HDFS_SERVER + Defines.HDFS_PATH
+ args[0]);
		String out = new String(in);
		String cb = new String(in);
		out += ".dir/compressed";
		cb += ".dir/codification";

		new Encoder(in, out, cb);

		in += ".dir/decompressed";
		new Decoder(out, in, cb);
	}
}
```

Encoder

```
package br.ufrj.ppgi.huffmanhadoop;

import java.io.IOException;
import java.util.Stack;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.util.ToolRunner;

import br.ufrj.ppgi.huffmanhadoop.mapreduce.EncoderConfiguration;
import br.ufrj.ppgi.huffmanhadoop.mapreduce.SymbolCountConfiguration;

public class Encoder {
	int[] frequency = new int[Defines.POWER_BITS_CODIFICATION];
	short symbols = 0;
	NodeArray nodeArray;
	Codification[] codification;

	public Encoder(String file_in, String file_out, String file_cb)
			throws Exception {
		String[] s = new String[3];
		s[0] = file_in;
		s[1] = file_out;
		s[2] = file_cb;
```

```
		/* MAPREDUCE SYMBOL COUNT */
		ToolRunner.run(new Configuration(), new SymbolCountConfiguration(),
s);
		/* END MAPREDUCE SYMBOL COUNT */
		fileToFrequency(file_in);
		frequencyToNodeArray();
		huffmanEncode();
		treeToCode();
		codificationToHDFS(file_cb);
		/* MAPREDUCE SYMBOL ENCODER */
		ToolRunner.run(new Configuration(), new EncoderConfiguration(), s);
		/* END MAPREDUCE SYMBOL ENCODER */
	}

	public void fileToFrequency(String arg0) throws IOException {
		Path path = new Path(arg0 + ".dir/symbolcount/");
		FileSystem fs = FileSystem.get(new Configuration());
		FileStatus[] status = fs.listStatus(path);

		for(byte i = 1 ; i < status.length ; i++) {
			FSDataInputStream f = fs.open(status[i].getPath());
			while(f.available() > 0) {
				byte symbol = f.readByte();
				frequency[symbol & 0xFF] = f.readInt();
				symbols++;
			}
		}

		frequency[Defines.EOF] = 1;
		symbols++;

		/*
		System.out.println("FREQUENCY: symbol (frequency)");
		for (int i = 0; i < frequency.length; i++)
			if (frequency[i] != 0)
				System.out.println((int) i + "(" + frequency[i] + ")");
		System.out.println("------------------------------");
		*/
	}

	public void frequencyToNodeArray() {
		nodeArray = new NodeArray((short) (symbols + 1));

		for (short i = 0; i < Defines.POWER_BITS_CODIFICATION; i++)
			if (frequency[i] > 0)
				nodeArray.insert(new Node((byte) i, frequency[i]));

		/*
		System.out.println(nodeArray.toString());
		*/
	}

	public void huffmanEncode() {
		while (nodeArray.size() > 1) {
```

```
                    Node a, b, c;
                    a = nodeArray.get(nodeArray.size() - 2);
                    b = nodeArray.get(nodeArray.size() - 1);
                    c = new Node((byte) 0, a.frequency + b.frequency, a, b);

                    nodeArray.removeLastTwoNodes();
                    nodeArray.insert(c);
                    /*
                    System.out.println(nodeArray.toString() + "\n");
                    */
            }
    }

    public void treeToCode() {
            Stack<Node> s = new Stack<Node>();
            codification = new Codification[symbols];
            Node n = nodeArray.get(0);
            short codes = 0;
            String path = new String();

            s.push(n);
            while (codes < symbols) {
                    if (n.left != null) {
                            if (!n.left.visited) {
                                    s.push(n);
                                    n.visited = true;
                                    n = n.left;
                                    path += "0";
                            } else if (!n.right.visited) {
                                    s.push(n);
                                    n.visited = true;
                                    n = n.right;
                                    path += "1";
                            } else {
                                    path = path.substring(0, path.length() - 1);
                                    n = s.pop();
                            }
                    } else {
                            n.visited = true;
                            codification[codes] = new Codification(n.symbol, path);
                            path = path.substring(0, path.length() - 1);
                            n = s.pop();
                            codes++;
                    }
            }

            /*
            System.out.println("CODIFICATION: symbol (size) code");
            for(short i = 0 ; i < symbols ; i++)
                    System.out.println(codification[i].toString());
            */
    }

    public void codificationToHDFS(String arg0) throws IOException {
```

```
                Path path = new Path(arg0);
                FileSystem fs = FileSystem.get(new Configuration());
                FSDataOutputStream f = fs.create(path);

                for (short i = 0; i < symbols; i++)
                        f.write(codification[i].toByteArray(), 0,
codification[i].toByteArray().length);
                f.close();
        }
}
```

Decoder

```
package br.ufrj.ppgi.huffmanhadoop;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FSDataOutputStream;
import org.apache.hadoop.fs.FileStatus;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;

import br.ufrj.ppgi.huffmanhadoop.Codification;
import br.ufrj.ppgi.huffmanhadoop.Defines;

public class Decoder {
        Codification[] codification;
        short symbols = 0;
        byte max_code = 0;
        Path in, out, cb;
        CodificationArrayElement[] treeArray;

        public Decoder(String file_in, String file_out, String file_cb)
                        throws IOException {
                in = new Path(file_in);
                out = new Path(file_out);
                cb = new Path(file_cb);

                fileToCodification();
                codeToTreeArray();
                huffmanDecode();
        }

        public void fileToCodification() throws IOException {
                FileSystem fs = FileSystem.get(new Configuration());
                FSDataInputStream f = fs.open(cb);

                codification = new
Codification[Defines.POWER_BITS_CODIFICATION];

                while (f.available() != 0) {
                        byte symbol = (byte) f.read();
                        byte size = (byte) f.read();
                        byte[] code = new byte[(size & 0xFF)];
```

```
                        f.read(code, 0, size & (0xFF));
                        codification[symbols] = new Codification(symbol, size, new
String(code));
                        if ((size & 0xFF) > max_code)
                                max_code = size;

                        symbols++;
                }

                /*
                System.out.println("CODIFICATION: symbol (size) code");
                for(short i = 0 ; i < symbols ; i++)
                        System.out.println(codification[i].toString());
                */
        }

        public void codeToTreeArray() {
                treeArray = new CodificationArrayElement[(int) Math.pow(2,
                        (max_code + 1))];

                for (short i = 0; i < symbols; i++) {
                        int index = 0;
                        for (char c : codification[i].code.toCharArray()) {
                                index <<= 1;
                                if (c == '0')
                                        index += 1;
                                else
                                        index += 2;
                        }
                        treeArray[index] = new CodificationArrayElement(
                                        codification[i].symbol);
                }

                /*
                System.out.println("codeToTreeArray():");
                System.out.println("TREE_ARRAY:");
                for(int i = 0 ; i < Math.pow(2,(max_code + 1)) ; i++)
                        if(treeArray[i] != null)
                                System.out.println("i: " + i + " -> " +
treeArray[i].symbol);
                System.out.println("-----------------------------");
                */
        }

        public void huffmanDecode() throws IOException {
                byte[] buffer = new byte[1];
                BitSet bufferBits = new BitSet();
                int index = 0;

                FileSystem fs = FileSystem.get(new Configuration());
                FileStatus[] status = fs.listStatus(in);
                FSDataOutputStream fout = fs.create(out);
```

```
                for (byte i = 1; i < status.length; i++) {
                //for (byte i = 1; i < 2; i++) {
                        FSDataInputStream fin = fs.open(status[i].getPath());
                        while (fin.available() > 3) {
                                fin.read(buffer, 0, 1);
                                bufferBits.fromByte(buffer[0]);
                                for (byte j = 0; j < Defines.BYTE_BIT; j++) {
                                        index <<= 1;
                                        if (bufferBits.cheackBit(j) == false)
                                                index += 1;
                                        else
                                                index += 2;

                                        if (treeArray[index] != null) {
                                                if (treeArray[index].symbol !=
Defines.EOF) {

        fout.write(treeArray[index].symbol);
                                        } else {
                                                index = 0;
                                                break;
                                        }
                                        index = 0;
                                }
                        }
                }
                        fin.close();
                }
                fout.close();
        }
}
```

Encoder Map

```
package br.ufrj.ppgi.huffmanhadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.BytesWritable;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Mapper;

import br.ufrj.ppgi.huffmanhadoop.Codification;
import br.ufrj.ppgi.huffmanhadoop.Defines;

public class EncoderMap extends
                Mapper<LongWritable, BytesWritable, LongWritable,
BytesWritableEncoder> {

        LongWritable key;
        int inc_key;
```

```
	short symbols = 0;
	Codification[] codification;

	@Override
	protected void setup(
			Mapper<LongWritable, BytesWritable, LongWritable,
BytesWritableEncoder>.Context context)
			throws IOException, InterruptedException {
		super.setup(context);

		key = new
LongWritable(context.getTaskAttemptID().getTaskID().getId());
		inc_key = context.getNumReduceTasks();

		codification = new
Codification[Defines.POWER_BITS_CODIFICATION];
		symbols = 0;

		String file_cb = context.getConfiguration().get("file_cb");
		Path path = new Path(file_cb);
		FileSystem fs = FileSystem.get(new Configuration());
		FSDataInputStream f = fs.open(path);

		while (f.available() != 0) {
			byte symbol = (byte) f.read();
			byte size = (byte) f.read();
			byte[] code = new byte[(size & 0xFF)];

			f.read(code, 0, size & (0xFF));

			codification[symbols] = new Codification(symbol, size, new
String(code));
			symbols++;
		}

		/*
		System.out.println("CODIFICATION: symbol (size) code");
		for(short i = 0 ; i < symbols ; i++)
			System.out.println(codification[i].toString());
		*/
	}

	public void map(LongWritable key, BytesWritable value, Context context)
			throws IOException, InterruptedException {
		BytesWritableEncoder buffer = new
BytesWritableEncoder(value.toString().length());

		for (int i = 0; i < value.getLength(); i++) {
			for (short j = 0; j < symbols; j++) {
				if (codification[j].symbol == value.getBytes()[i]) {
					buffer.addCode(codification[j]);
					break;
				}
```

```
                        }
                }
                context.write(this.key, buffer);
                this.key.set(this.key.get() + this.inc_key);
        }
}
```

Encoder Reduce

```
package br.ufrj.ppgi.huffmanhadoop.mapreduce;

import java.io.IOException;

import org.apache.hadoop.conf.Configuration;
import org.apache.hadoop.fs.FSDataInputStream;
import org.apache.hadoop.fs.FileSystem;
import org.apache.hadoop.fs.Path;
import org.apache.hadoop.io.LongWritable;
import org.apache.hadoop.mapreduce.Reducer;

import br.ufrj.ppgi.huffmanhadoop.Codification;
import br.ufrj.ppgi.huffmanhadoop.Defines;

public class EncoderReduce extends Reducer<LongWritable, BytesWritableEncoder,
LongWritable, BytesWritableEncoder> {
        private static final int INITIAL_CAPACITY = 100;
        private BytesWritableEncoder bw;

        Codification eof;

        @Override
        protected void cleanup(
                        Reducer<LongWritable, BytesWritableEncoder, LongWritable,
BytesWritableEncoder>.Context context)
                        throws IOException, InterruptedException {
                bw.addCode(eof);
                context.write(new LongWritable(0), bw);
                super.cleanup(context);
        }

        @Override
        protected void setup(
                        Reducer<LongWritable, BytesWritableEncoder, LongWritable,
BytesWritableEncoder>.Context context)
                        throws IOException, InterruptedException {
                super.setup(context);

                String file_cb = context.getConfiguration().get("file_cb");
                Path path = new Path(file_cb);
                FileSystem fs = FileSystem.get(new Configuration());
                FSDataInputStream f = fs.open(path);

                while (f.available() != 0) {
                        byte symbol = (byte) f.read();
                        byte size = (byte) f.read();
                        byte[] code = new byte[(size & 0xFF)];
```

```
                        if(symbol != Defines.EOF)
                                f.skip(size & (0xFF));
                        else {
                                f.read(code, 0, size & (0xFF));
                                eof = new Codification(symbol, size, new String(code));
                        }
                }
                bw = new BytesWritableEncoder(INITIAL_CAPACITY);
        }

        public void reduce(LongWritable key, Iterable<BytesWritableEncoder> values,
        Context context)
                        throws IOException, InterruptedException {
                for (BytesWritableEncoder b : values)
                        bw.addBytesWritable(b);
        }
}
```

EK 9

Hadoop Sıkıştırma Kod Bloğu

Sıkıştırma Örneği

```
public static CompressionCodec getCompressionCodecProperty(Properties p , String key)
{
	String value = getStringProperty(p, key).toLowerCase();

	if (value.equals("snappy"))
	{
		return new SnappyCodec();
	}else if (value.equals("gzip"))
	{
		return new GzipCodec();
	}else if (value.equals("bzip2"))
	{
		return new BZip2Codec();
	}else
	{
		return new SnappyCodec();
	}
}
```

EK 10

MongoDB Kodları

Veri Ekleme Kod Bloğu

```
package mongo.newrecord;

import java.io.File;
import java.io.IOException;
import java.net.UnknownHostException;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSInputFile;

public class App
{
   public static void main( String[] args )
   {
            System.out.println( "bağlanıyor..." );
       try {

            MongoClient mongoClient = new MongoClient("localhost", 27017);

            DB db = mongoClient.getDB("PACS");
            System.out.println("Collection selected successfully");


                              String name = "";
                              for(int i= 0 ; i<100 ; i++)
                              {
                                       name = "Image_" + i;

                                       DBCollection col = db.getCollection("image");
                     String newFileName = name;


                     String filePathStr = "/home/erdal/Masaüstü/PACS.png";
                                       File filepath = new File(filePathStr);



                              // create a "photo" namespace
                                       GridFS gridfile = new GridFS(db, name);

                                       // get image file from local drive
                              GridFSInputFile gfsFile = gridfile.createFile(filepath);
                    // set a new filename for identify purpose
                              gfsFile.setFilename(newFileName);

                              // save the image file into mongoDB
                              gfsFile.save();

                              // print the result
```

```
                        DBCursor cursor = gridfile.getFileList();

                        while (cursor.hasNext()) {
                                System.out.println(cursor.next());
                        }
                        }









                        // get image file by it's filename
                //      GridFSDBFile forOutput = gridfile.findOne(newFileName);
                //      forOutput.writeTo("/home/erdal/new.png");

                        System.out.println("Done");

                } catch (UnknownHostException e) {
                        e.printStackTrace();
                } catch (MongoException e) {
                        e.printStackTrace();
                } catch (IOException e) {
                        e.printStackTrace();
                }
    }
}
```

Veri Çekme Kod Bloğu

```
package mongo.newrecord;

import java.io.File;
import java.io.IOException;
import java.net.UnknownHostException;

import com.mongodb.BasicDBObject;
import com.mongodb.DB;
import com.mongodb.DBCollection;
import com.mongodb.DBCursor;
import com.mongodb.MongoClient;
import com.mongodb.MongoException;
import com.mongodb.gridfs.GridFS;
import com.mongodb.gridfs.GridFSDBFile;
import com.mongodb.gridfs.GridFSInputFile;

/**
 * Hello world!
 *
 */
public class App
{
    public static void main( String[] args )
    {
            System.out.println( "bağlanıyor..." );
        try {
            MongoClient mongoClient = new MongoClient("localhost", 27017);
           DB db = mongoClient.getDB("PACS");
           System.out.println("Collection selected successfully");
```

```
            String newFileName = "Image_11";
              GridFS gfsPhoto = new GridFS(db, newFileName);  //ns name
              GridFSDBFile imageForOutput = gfsPhoto.findOne(newFileName);
              System.out.println(imageForOutput);
              imageForOutput.writeTo("/home/erdal/Masaüstü/PACS_GeneratedEE.png");


                        } catch (UnknownHostException e) {
                                e.printStackTrace();
                        } catch (MongoException e) {
                                e.printStackTrace();
                        } catch (IOException e) {
                                e.printStackTrace();
                        }
    }
}
```

ÖZGEÇMİŞ



Adı Soyadı : Erdal ERDAL
Doğum Tarihi : 12.11.1986
Yabancı Dil : İngilizce

Eğitim Durumu

Lisans : Çankaya Üniversitesi Mühendislik Fakültesi Bilgisayar Mühendisliği 2005-2010

Yüksek Lisans : Bahçeşehir Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği 2011-2013

Doktora : Kırıkkale Üniversitesi Fen Bilimleri Enstitüsü Bilgisayar Mühendisliği 2013-2017

Çalıştığı Kurum/Kurumlar ve Yıl/Yıllar:

Entegre Enformasyon Sistemler	2010-2011
Koç Sistem Bilgi ve İletişim Hizmetleri	2011-2013
Kırıkkale Üniversitesi	2013-

Yayınları (SCI) :

A. Ergüzen ve E. Erdal, «Medical Image Archiving System Implementation with Lossless Region of Interest and Optical Character Recognition,» Journal of Medical Imaging and Health Informatics, 2017. (Accepted Manuscript)

Yayınları (Diğer) :

E. Erdal ve A. Ergüzen, «The New Era of Medical Imaging: Cloud, » International Journal of Advances in Electronics and Computer Science, Cilt 4, No. 2, 2017.

E. Erdal ve A. Ergüzen, «The New Era Of Medical Imagıng: Cloud,» 115th International Conference On Science, Technology And Management (ICSTM), Florence, Italy, 28th-29th December, 2016.

E. Erdal ve A. Ergüzen, «A Parallel Approach For Determining Region–Of–Interest Area Of Digital Medical Images,» 28th International Conferences on Engineering and Natural Science (ICENS), Amsterdam, Netherland, 2015.

E. Erdal, Y. B. Salman ve H. I. Cheng, «Real Time Information Assistant for Blind People using the Internet,» 17th International Conference on Industrial Engineering: Theory, Applications and Practice Pusan National University., Busan, Korea, 2013.

E. Erdal ve A. Ergüzen, «Kırıkkale Üniversitesi Personel Portal Sistemi ve Öğrenci Portal Sistemi Geliştirilmesi ve Getirdikleri,» 19. Akademik Bilişim, Aksaray, Turkey, 2017.

Araştırma Alanları :

Bilgisayar yazılımı, veritabanı sistemleri, dağıtık sistemler, algoritmalar, yazılım geliştirme.