İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

IOS TABANLI MOBİL CİHAZLAR İÇİN YAPAY ZEKA YÖNTEMİ İLE GÖRÜNTÜ SIKIŞTIRMA VE İLETİM UYGULAMASI

YÜKSEK LİSANS TEZİ

Ömer GÜNGÖR

Sistem Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Ayşegül ALAYBEYOĞLU

TEMMUZ 2017

İZMİR KÂTİP ÇELEBİ ÜNİVERSİTESİ ★ FEN BİLİMLERİ ENSTİTÜSÜ

IOS TABANLI MOBİL CİHAZLAR İÇİN YAPAY ZEKA YÖNTEMİ İLE GÖRÜNTÜ SIKIŞTIRMA VE İLETİM UYGULAMASI

YÜKSEK LİSANS TEZİ

Ömer GÜNGÖR

(600113011)

Sistem Mühendisliği Anabilim Dalı

Tez Danışmanı: Doç. Dr. Ayşegül ALAYBEYOĞLU

TEMMUZ 2017

İKÇÜ, Fen Bilimleri Enstitüsü'nün 600113011 numaralı Yüksek Lisans Öğrencisi Ömer GÜNGÖR, ilgili yönetmeliklerin belirlediği gerekli tüm şartları yerine getirdikten sonra hazırladığı "IOS TABANLI MOBİL CİHAZLAR İÇİN YAPAY ZEKA YÖNTEMİ İLE GÖRÜNTÜ SIKIŞTIRMA VE İLETİM UYGULAMASI" başlıklı tezini aşağıda imzaları olan jüri önünde başarı ile sunmuştur.

Tez Danışmanı :	**Doç. Dr. Ayşegül ALAYBEYOĞLU** İzmir Katip Çelebi Üniversitesi	.....
Jüri Üyeleri :	**Yrd. Doç. Dr. Kökten Ulaş BİRANT** Dokuz Eylül Üniversitesi	.....
	Doç. Dr. Femin YALÇIN İzmir Katip Çelebi Üniversitesi	.....

Teslim Tarihi : 12 Haziran 2017
Savunma Tarihi : 10 Temmuz 2017

Çok Değerli Eşime,

ÖNSÖZ

Yapay sinir ağları ve görüntü işleme teknikleri günümüzde birçok alanda kullanılmaktadır. Bu çalışmada günlük hayatta çok kullandığımız bir uygulamaya farklı bir bakış açısı getirilerek bir uyarlama oluşturmaya çalışıldı. İnsanlar çevrelerindeki diğer insanlarla sürekli bir dosya paylaşım halindedirler. Teknolojinin ilerlemesi ile birlikte bu paylaşımlar çoğunlukla büyük boyutlu veriler olan fotoğraf ve video paylaşımları olmaktadır. Neredeyse bazı insanların her anı artık dijital platformlarda kayıtlı haldedir. Durum bu şekilde olunca, gün geçtikçe daha çok enerji, daha çok saklama alanı, daha çok bant genişliği gerekmektedir.

Tez kapsamında yapılan çalışma ile bu kaynak kullanımını düşürebilmek ve enerjiyi verimli kullanabilmek amacıyla iki farklı yapay zekâ yöntemi kullanılarak görüntü sıkıştırma ve gönderimi gerçekleştirilmiştir. İki farklı algoritma arasında hangisi daha az enerji harcıyor, hangisi daha az yer kaplıyor test sonuçlarında görülmüştür.

Bu çalışmayı hazırlarken sağlamış olduğu cihaz desteğinden dolayı İzmir Katip Çelebi Üniversitesi'ne teşekkür eder, geçirdiğim süre zarfında benden desteğini eksik etmeyen Doç. Dr. Ayşegül ALAYBEYOĞLU'na sonsuz teşekkürlerimi ve saygılarımı sunarım.

Tüm eğitim hayatım boyunca bana yol gösteren, destek olan, en önemlisi bana eğitimin ne kadar önemli olduğu bilincini ve çalışma disiplinimi kazandıran, her zaman yanımda olan Dr. Çağdaş Can BİRANT'a en içten dileklerimle teşekkürlerimi sunuyorum.

Beni bu günlere getiren onlar için bir gurur kaynağı olmamı sağlayan, ülkeye faydalı bir birey olmamı sağlayan aileme çok teşekkür ediyorum.

Ayrıca bu tezde beni destekleyen, yardımcı olan tüm dostlarıma teşekkürü bir borç bilirim.

TEMMUZ 2017

Ömer GÜNGÖR

IKCU Bilgisayar
Mühendisliği Bölümü

Yüksek Lisans Öğrencisi

İÇİNDEKİLER

SAYFA

ŞEKİL LİSTESİ

SAYFA

IOS TABANLI MOBİL CİHAZLAR İÇİN YAPAY ZEKA YÖNTEMİ İLE GÖRÜNTÜ SIKIŞTIRMA VE İLETİM UYGULAMASI

ÖZET

Bilgi çağında iletişim teknolojileri hızla gelişmektedir. İnsanlar, aralarındaki iletişimi arttırmak için teknolojiden yararlanmaktadırlar. Bu durum bilginin iletilmesi ve iletilirken kaynakların verimli kullanılmasını zorunlu kılmıştır. Bu kaynaklar enerji, saklama alanı, bant genişliği gibi sürekli kullanılan kaynaklardır.

Bu tez kapsamında yapılan çalışmanın amacı IOS işletim sistemi kullanan cihazlarda görüntüyü iki farklı yapay zekâ algoritması ile sıkıştırarak görüntünün boyutunun düşürülerek enerji tüketiminden kazanç sağlanması ve performans analizlerinin incelenmesidir.

IOS İşletim sistemine sahip olan cihazlar günümüzde oldukça popülerdir. En büyük rakibi olan Android işletim sistemine sahip cihazların maliyet olarak daha uygun olmasından ve Android Market'in bir firmaya değil de bir topluluğa aitmiş gibi olmasından dolayı günden güne IOS tarafında popülarite düşse de, aralarında büyük bir rekabet vardır. Bu rekabet ve IOS işletim sistemine sahip cihazların çok satılması bu konuda araştırma yapmak için yeterince büyük bir sebep teşkil etmektedir.

Tez araştırma aşamasında IOS İşletim sistemi cihazlarda kullanılabilen diğer mesajlaşma programları ve diğer işletim sistemlerinde kullanılan mesajlaşma programları incelenmiştir. Görüntü transferlerinde daha az kaynak harcamak, teknolojiyi verimli kullanmak için yapay zekâ ve görüntü sıkıştırma yöntemleri kullanılmıştır.

IMAGE COMPRESSION AND TRANSMISSION APPLICATION FOR IOS BASED MOBILE DEVICES WITH ARTIFICIAL INTELLIGENCE

ABSTRACT

In recent years, Communication technologies are growing rapidly. Developments in technology is very important for increasing the communication between individuals. This situation requires efficient transmission of information and mean while efficient usage of communication resources such as energy and storage.

The aim of this study is to compress the image with two different artificial intelligence algorithms and transmit it to the others by using devices which is operating by IOS operating system.

Nowdays, devices using the IOS operating system are much popular. There is still huge competition between IOS devices and Android which lost interest day by day and the Android devices are even the biggest competitor Android OS has less device's cost and also it’s like belong to public not a company. That competition and huge sales volume of IOS devices in use forces to study about this work for enough reason.

All other messaging applications which work on IOS and Android are analyzed while preparing dissertation. Artificial intelligence and image compression methods were used for the less source consumption and the high efficiency while using technology.

1. GİRİŞ

1990 yılından sonra uluslararası bir ağ haline gelen internet, günümüzde modern insan yaşamının vazgeçilemez bir unsuru haline gelmiştir. Gelişen teknoloji, düşen maliyetler, bilgiye ulaşma, paylaşma ve haberleşme ihtiyacının giderek artması ile hızla yayılmıştır.

İstatiksel olarak internetin yayılma hızı ilk 50 milyon insan için sadece 5 yıl sürmüştür. Bu rakam radyo için 38 yıl, televizyon için 13 yıl sürmüştür[35]. 21. yüzyılın getirdiği mobilleşme süreciyle internet her yerden her an erişebilir bir duruma gelmiştir. Bu da insanlığın başlıca ihtiyaçlarından biri olan iletişim kurmayı, sohbet etmeyi, mesajlaşmayı hayatın her anına yaymış ve kısa bir süre içinde etkileşimi küresel boyuta getirmiştir. Tüm bu artan mesajlaşma isteği kişisel ve kurumsal bilginin hızlı ve güvenli bir şekilde iletilmesini gerektirmiştir. Mobil cihazların günün her saatinde ihtiyaç dâhilinde veya haricinde kullanılması, görsel içerik bulunan anlık mesajların büyüklüğü gibi durumlar neticesinde giderek artan veri depolama ve enerji ihtiyacını doğurmuştur. Bu ihtiyacı azaltacak başlıca çözüm ise, yazılım ile gerçekleştirilebilecek veri sıkıştırma teknikleri olmuştur. Bu çalışmanın amacı, IOS işletim sistemi ile çalışan cihazlarda sohbet etmek ve aynı ortamda sohbet edilen kişilerle görüntü paylaşımı yaparken, görüntüyü yapay zekâ yöntemleri ile sıkıştırarak göndermektedir. Ayrıca kullanılan yapay zekâ algoritmalarının performanslarının karşılaştırılması da bu çalışmanın amaçları arasında yer almaktadır.

IPhone ve IPad cihazlar IOS işletim sistemi kullanmaktadırlar ve bu cihazların sayısı her geçen gün artmaktadır. Bu cihazlar, hafızalarındaki kendi oluşturdukları yüksek çözünürlüklü görüntüleri, sıkıştırmadan olduğu gibi göndermekte olup, yüksek bant genişliği ve harcanan zamanla doğru orantılı olarak fazla enerji tüketimi gerçekleştirmektedirler. İşte bu noktada, bu tez çalışması çözüm olarak yapay zekâ ile görüntüleri sıkıştırarak, harcanan fazla enerjinin azaltılmasını öngörmektedir. Yapay zekâ ile sıkıştırılarak gönderilmekte olan veri, mümkün olan en iyi algoritmayı seçerek veriyi sıkıştırmaktadır. Bunun sonucunda veri hızlı ve güvenli bir şekilde alıcı veya alıcılara ulaştırılmaktadır.

Tezin 2. Bölümünde yapay zekâ ile görüntü sıkıştırma konusu ve benzer konularda literatür araştırması yapılmış, bulunan araştırmalarla ilgili kısa genel bilgiler verilmiştir. 3. Bölümünde çalışmanın altyapısını oluşturan yapay zeka, görüntü sıkıştırma, IOS programlama, K-Means algoritması, KNN algoritması, Delta sıkıştırma algoritması, Socket.IO programlama konuları hakkında bilgiler verilmiştir. 4. Bölümde teze çalışmasına konu olan IOS uygulaması hakkında detaylı tüm bilgiler verilmiştir. 5. Bölümde ortaya çıkarılan ürün çeşitli performans testlerine dâhil edilerek, bu testlerin sonuçları grafikler halinde gösterilmiştir. 6. Bölümde tezin amacı dâhilinde ulaşılan sonuç ve gelecekte bu çalışmanın gelişmesi için yapılabilmesi muhtemel öneriler verilmiştir.

2. İLGİLİ ÇALIŞMALAR

Bu konuda yapılan araştırmalar arasında IOS işletim sistemi kullanılan cihazlarda yapay zekâ algoritması kullanılarak görüntü sıkıştırma ile ilgili yapılan araştırmaya rastlanamamıştır. Fakat sadece yapay zekâ ile görüntü sıkıştırma veya görüntü sıkıştırma ile ilgili araştırmalar literatürde mevcuttur.[1]'de internetin yaygınlaşmasıyla dijital görüntü paylaşımının arttığına dikkat çekilmiştir. Görüntülerin sıkıştırılarak ve güvenli bir şekilde aktarılması sağlanmıştır. Güvenlik altyapısı sayısal güvenlik sertifikalarıyla sağlanmış olup sıkıştırma işlemlerinde de temel bileşenler analizi ve dalgacık tabanlı dönüşüm teknikleri kullanılmıştır.[2]'de yapay sinir ağlarının dijital görüntü sıkıştırılmasında kullanımı araştırılmıştır. Bant genişliğinin fazla kullanıldığına dikkat çekerek, görüntü sıkıştırmayı kullanıp gereksiz kullanılan bant genişliğinin azaltılması hedeflenmiştir. Dönüşüm tabanlı yöntemlerde, sıkıştırılacak görüntü, üst üste binmeyen n x n piksel boyutlarında bloklara ayrıştırılarak kullanılmıştır. N boyutlu uzayda her blok N boyutlu bir vektör olarak düşünülmüştür. [3]'te yapay sinir ağları ile görüntü sıkıştırma ve görüntü kütük biçimi ile ilgili bir araştırma yapılmıştır. Bu araştırmada insandaki görme duyusunun yapay zekâ ile taklit edebilme yetisi amaçlanmıştır. Görme duyusu taklit edilerek sayısal görüntü oluşturulup, bu oluşan görüntün sahip olduğu, insan gözünün göremeyeceği kısımlar temizlenerek sıkıştırılmış bir resim oluşturulmaya çalışılmıştır. Bu işlem yapılırken ileri beslemeli yapay sinir ağı modeli ve geri yayılımlı öğrenme kullanılmıştır.[4]'te yapay sinir ağları kullanılarak görüntü sıkıştırma konusu araştırılmıştır. Bu çalışma, sayısal renkli görüntülerin sıkıştırılması için iki yöntem karıştırılarak yeni bir teknik önerilmiştir. Bu çalışmada dalgacık dönüşümü ve yapay sinir ağları birlikte kullanılmıştır. Sıkıştırma değerleri diğer genel geçer algoritmalarla karşılaştırıldığında ciddi bir olumsuz performans farkı görülmemiştir. Aksine bu algoritmayla yapılan testlerde yüksek Bit/Pixel değerlerinde, bazı test görüntülerinde, diğer algoritmaların mantığına göre daha yüksek sıkıştırma performansları bulunmuştur. [5]'te öğrenebilen durum makinesi yaklaşımına dikkat çekilmiştir. Öğrenebilen durum makineleri için bir simülatör yapılmıştır. Amaç ise bu öğrenebilen durum makinelerini yaygınlaştırmaktır. Böylece kullanıcılar kendi uygulamalarını bu makinelerde deneme fırsatı yakalayacaklardır. Diğer amaç ise ağlarda kayıpsız sıkıştırma için tek ölçü birimi olan sıkıştırma oranını geliştirmektir. [6]'da fraktal görüntü sıkıştırma üzerinde

durulmuştur. Tekrarlamalı fonksiyon sistemlerinin teorisi üzerine kurulu olan fraktal görüntü sıkıştırmasının temel görüşleri tanıtılmıştır. Ayrıca fraktal algoritması kenar tespitinde kullanılmıştır. [7]'de bilinen ve kullanılan en yaygın sıkıştırma biçimi olan jpegden yola çıkılarak, jpegden iki kat daha iyi sıkıştırma oranına sahip ve işlem karmaşıklığı daha az olan jpegxr incelenmiştir. Jpeg ve jpegxr standartları incelenmiş ve arasındaki fark nesnel olarak ortaya koyulmuştur. [8]'de spesifik olarak bir algoritma veya bir uygulama değilde günümüzde en çok kullanılan görüntü sıkıştırma teknikleri hakkında bilgi verilmiştir. Üzerinde durulan algoritmalar run-length coding, ayrık fourier, kosinüs transform ve vektör kuantalamadır. Sonuç olarak vektör kuantalamanın diğerlerine göre daha yüksek sıkıştırma oranının olduğu görülmüştür. [9]'da Biyomedikal görüntülerin dalgacık dönüşümü ile sıkıştırılması konusu araştırılmıştır. Bu çalışmada biyomedikal görüntüleri sıkıştırmak için dalgacık dönüşümü algoritması kullanılmıştır. Bir mr, bir mamogram ve birde ultrasonik biyomedikal görüntüler ile lena görüntüsü dalgacık kodlayıcı ile sıkıştırılmış ve sonuçları karşılaştırılmıştır. [10]'da görüntü sıkıştırmak için dalgacık dönüşümü yöntemi seçilmiştir. Dalgacık dönüşümünde oluşan alt ağaçlar sayesinde küme bölümleme algoritması ile birlikte görüntüde yüksek sıkıştırma sağlanması amaçlanmıştır. [11]'de araştırmacı türevsel bilgi kodlamalı modülasyon(TBKM) üzerinde durmuş. Görüntü sıkıştırma, tahmin edilebilir düzenleme, doğrusal yaklaşım, türevsel bilgi kodlamalı modülasyon (TBKM), çıkışı otoregresif, kayan ortalamalı otoregresif uyarlanabilir katsayılı TBKM teknikleri ile gerçekleştirilmiştir. Sonuçlar algoritmanın performansı, ortalama mutlak hata ve sinyalin gürültüye oranı dikkate alınarak bulunmuştur. En başarılı sonuçların görüntü elemanının 1 bit oranında gösteriminde otoregresif kayan ortalamalı çıkış veren TBKM tekniği olduğu gözlenmiştir. [12]'de fraktal görüntü sıkıştırma yönteminin günümüzdeki ilgi gören sıkıştırma yöntemleri kadar önemli bir yöntem olduğu anlatılmaktadır. Formülsel veri çok bulunmamakla birlikte, Fractal görüntü sıkıştırma hakkında genel bilgi ve matematiksel çalışma içermektedir. [13]'te parmak izi görüntülerinin kayıpsız olarak insan gözü için değilde tanıma yazılımları için gerekli düzeyde sıkıştırılması araştırılmıştır. Normalde algoritmaların insan gözünün göremeyeceği bilgileri ortadan kaldırarak sıkıştırma yoluna gittiği anlatılmıştır. Fakat konu parmak izi tanıma olunca her bilgi ayırt edici unsur olduğundan bilgi silmeden veri sıkıştırma araştırılmıştır. [14]'te araştırmacı kayıplı ve kayıpsız veri sıkıştırma teknikleri üstünde durmuştur. Minimum fazlalık kodlama, aritmetik kodlama ve run-length

kodlama teknikleri incelenmiştir. Ayrıca Geri yansımalı yapay sinir ağları üzerinde durulmuş ve bu konuda algoritma uygulanmıştır. Sonuç olarak %25 sıkıştırma ve %3 hata ile araştırma sonuçlanmıştır. [15]'te araştırmacı radyal tabanlı fonksiyon üzerinde durmuştur. Radyal taban fonksiyonlu yapay sinir ağının fraktal görüntü sıkıştırmada görüntü kalitesi ve sıkıştırma oranını arttırmak amacıyla kullanılabilirliği araştırılmıştır. Geliştirilen yöntem ile görüntü bloklarının alt bloklara dönüştürüldüğü sabit bölmelemeli dönüşüm yöntemini temel almıştır. Bilinen fraktal kodlama yordamlarından farkı, doğrusal büzülme dönüşümleri yerine radyal tabanlı yordamların kullanılmasıdır. [16]'da veri sıkıştırma yöntemlerini karşılaştıran bir çalışma yapılmıştır. Uygulamada en çok kullanılan görüntü sıkıştırma teknikleri incelenerek görüntüler üzerindeki farkları incelenmiştir. Run-length kodlama, ayrık fourier transform, ayrık kosinüs transform, vektör kuantalama, hiyerarşik vektör kuantalama, wavelet transform teknikleri karşılaştırılıp, gürültü oranları ve mutlak hata oranları bulunmuştur. [17]'de kayıplı görüntü sıkıştırma yöntemlerinden DCT temelli görüntü sıkıştırma yöntemi incelenmiştir. DCT katsayılarını alt bantlarına ayırmak ve bu alt-bantlara bit tahsisi, sıfır atama, kuantalama işlemlerinin alt-bantlara dayandırılarak işlenmesinin, kodlama kalitesi ve sıkıştırma oranının arttırılması amacıyla incelenmiştir. DCT matrisinin boyutunu, kesme faktörünü ve alt-bantlara tahsis edilecek bit miktarlarını parametre dosyasından okuyarak sıkıştırma işlemi gerçekleştirilmiştir. [18]'de araştırmacı Matlab kullanarak birden fazla görüntü sıkıştırma tekniklerini karşılaştırmıştır. Karşılaştırdığı teknikler huffman kodlama, wavelet ve jpeg teknikleridir. Sıkıştırma oranı ile kayıpların nasıl etkilendiği ölçülmüştür.[19]'da dalgacık dönüşümü fonksiyonu araştırılmış ve uygulanmıştır. Sonuç olarak ise algoritmalar arası görüntü kalitesine göre karşılaştırma yapılmıştır.[20]'de kayıplı ve kayıpsız veri sıkıştırma teknikleri araştırılmıştır. Minimum fazlalık kodlama, aritmetik kodlama ve run-length kodlama teknikleri incelenmiştir. Görüntü Sıkıştırma kodlayıcılarından vektör gruplama yöntemi üzerinde durulmuştur. Yapay sinir ağları ile görüntü sıkıştırma teknikleri incelendiğinde ise çok tabakalı algılayıcı ile görüntü sıkıştırma ile ilgili bilgiler verilmiş ve algoritma uygulanmıştır. İnsanın görme sistemi fonksiyonları ile birlikte renkli resim işlemenin teorik temelleri de araştırmada yer almaktadır. [21]' de görüntü sıkıştırma yöntemlerinden DCT (DiscreteCosineTransform-Ayrık Cosinüs Dönüşümü) ve DWT (DiscreteWalshTransform- Ayrık Walsh Dönüşümü) 'nin karşılaştırılması yapılmıştır. DCT ve DWT tekniklerini karşılaştırmak için

sıradan görüntüler kullanılmıştır. Borland Delphi programlama dilinde yazılmış program üstünde denemeler yapılmıştır. İşlenecek görüntü yüklemiştir, daha sonraki aşamada girilen boyut, ihmal ve üst değerlerine bağlı olarak görüntü sıkıştırılmıştır. Her iki yöntem test edildikten ve işlendikten sonra iki dönüşüm arasındaki farklar sonuç olarak gösterilmiştir. [22]'de gri seviyeli manzara görüntülerine sıkıştırma amaçlı kullanılabileceğinin önerisi yapılmıştır.İki yöntem kullanılmıştır. Fonksiyon bazlı bölütleme ve bulanık mantık bazlı bölütleme. Yapılan işlem, bulanık mantık temelli görüntü bölütleme veya bulanık görüntü bölütlemedir. Araştırma sonucunda bölge büyütme ile bölütlenen görüntülerde çevrit veya zincir kodlama (chain coding) daha iyi sonuçlar vermiştir. [23]'de kablosuz sensörler ile kullanılabilen dağıtılmış görüntü sıkıştırma araştırması yapılmıştır. Sistemin dağıtılmış görüntü sıkıştırma adımlarını ve dinamik küme yapısına odaklanılmıştır. İki durumun çalışma prensiplerini gösteren diyagramlar ve çizelgeler sunulmuştur. [24]'de mobil kablosuz multimedya sensörler için genetik algoritma tabanlı kümeleme ile görüntü sıkıştırma araştırılmıştır. Genetik algoritma tabanlı kümeleme ile kümelerin tekdüzeliği, enerji tüketimi, baz istasyonuna olan uzaklığı dikkate alınarak geliştirilmiştir.

3. ÇALIŞMANIN ALTYAPISINI OLUŞTURAN KONULAR

Bu araştırmayı oluşturan birbirinden faklı altyapılar vardır. Bu alt yapıların neler oldukları, hangi konularla alakalı oldukları hakkında genel bilgiler verilmiştir.

3.1. Yapay Zekâ

Yapay zekâ algılama, öğrenme, çoğul kavramları bağlama, düşünme, fikir yürütme, sorun çözme, iletişim kurma, çıkarım yapma ve karar verme gibi işleri yapabilmesi beklenen yapay bir işletim sistemidir [25].

Başlangıçta bir bakış açısına yönelik geliştirilse de günümüzde bu düşünce değişmektedir. Çünkü artık birçok problem yapay zekâ ile çözümlenebilmektedir. Gelecekte insan zekâsından bağımsız olarak çalışabilecek bir yapay zekâ kuramına doğru yeni yönelimler oluşmaktadır. Bu yönelim ilk olarak Karel Čapek tarafından, R.U.R adlı tiyatro oyununda yapay zekâya sahip robotlar ile insanlığın toplumsal sorunlarını ele alarak 1920 yılında yapay zekânın insan zekâsından bağımsız çalışabileceğini öngörmüştür[26].

3.2. Görüntü Sıkıştırma

Dijital görüntünün birçok yöntemi 1960'lı yıllarda Jet Propulsion Laboratuvarı, Massachusetts Teknoloji Enstitüsü gibi araştırma tesislerinde; görüntülü konuşma, tıbbi görüntüleme, karakter tanıma, iyileştirme gibi uygulamalar için geliştirilmiştir [27]. Fakat dönemin şartlarında bu tarz işlemler oldukça maliyetli olduğu için bilgisayarların çoğalmasına kadar kolay olarak yapılamamıştır. Kişisel bilgisayarlar daha performanslı hale geldiğinden bu yana sayısal görüntü işleme alanındaki işlemler daha kolay ve az maliyetli hale gelmiştir. Dijital teknolojinin de gelişmesiyle birlikte analog görüntüyü dijital olarak ifade edebilmek için çözünürlük kavramına ihtiyaç duyulmuştur. Gerçek görüntüyü dijital olarak ifade ederken görüntünün noktalardan oluşturulabileceği fikrinden hareket edilmiştir. Bu noktalar ise piksel olarak adlandırıldı. Bir dijital görüntüde ne kadar çok nokta var ise o kadar kaliteli ve gerçeğe yakın bir sonuç elde edilir. Görüntüyü gerçeğe yakınlaştıran bir diğer etken ise görüntüyü oluşturan her bir noktanın kendi içindeki derinliğidir. Görüntüyü oluşturan her bir pikselin renk aralığı ne kadar fazla olursa,

piksel bulunduğu küme içinde o kadar gerçeğe yakın bir renk verecektir. Bu olaya renk derinliği denir. Örnek olarak 1 bit renk derinliğine sahip bir pikselde yalnızca 2^1 = 2 adet renk bulunmaktadır. Bu renkler siyah ve beyazdır. Günümüzde kullanılan monitörler, ekranlar, çekilen videolar, resimler genelde 32bit renk derinliği kullanılarak oluşturulur. 32 bir renk derinliğine sahip bir piksel ise 2^{32} = 4.294.967.296 renk alabilir. Buda renk derinliğinin gerçeğe ne kadar yaklaştırdığının kanıtıdır.

Günümüzde renklerin tamamını Kırmızı-Yeşil-Mavi(RGB) renkleri karıştırılarak bulunabilir. Şekil 3.1'de RGB renk uzayını temsil eden bir resim görülüyor. RGB renk uzayında gerçek renk yani 32 bit renk üretildiğinde aslında 4.294.967.296 renk değil 16.777.216(2^24) renk gösterebilir. Bunun sebebi RGB renk uzayı 32 bitlik veriyi 4e böler ve bunun 8 bitlik kısmını Kırmızıya, 8 bitlik kısmını Yeşile, 8 Bitlik kısmını Maviye ve son 8 Bitlik kısmını ise Alfa değerine ayırır. Alfa değerinin renk ile bir alakası yoktur sadece gösterilen rengin şeffaflık değerini belirtir. [35]

Şekil 1.1: RGB renk uzayı.

Bir görüntünün çözünürlüğü yataya düşen nokta sayısı ile düşeye düşen nokta sayınının çarpımıyla bulunur. A x B ile gösterilir. Örneğin: 1024 x 768 çözünürlükteki bir resimde 786.432 adet nokta var anlamına gelir. Yani 786.432 adet nokta bize 1024 x 768 çözünürlüğünde bir görüntü oluşturur. [35]

Bir görüntünün boyutu düşey nokta sayısı x yatay nokta sayısı x renk derinliğin olarak hesaplanır. Örneğin 32 bit gerçek renk ve yine 1024 x 768 çözünürlüğünde bir resim ele alınırsa, 1024 x 768 x 32 = 25.165.824 bit = 3.145.728 byte = 3.072 kilobyte = 3 megabyte boyutunda bir resim elde edilmiş olur. [35]

Günümüzde kullanılan cihazlar ele alındığında çekilen fotoğrafların kalitesi çok yüksek olduğundan dolayı verilen örnek aslında çok küçük boyutlarda kalmıştır. Bu

sebeple görüntüleri sıkıştırmadan göndermek veya işlemek enerji ve kaynak maliyetini arttırır. Görüntü sıkıştırma yöntemleri kullanılarak sıkıştırılan bir resmin indirilmesi, iletilmesi normale göre daha az enerjiye, bant genişliğine ve kaynağa ihtiyaç duyar.

Görüntü sıkıştırmanın temelinde birbirine ilişkisi aynı olan komşu pikselleri temizlemek veya aza indirgemek vardır. Bunun sebebi ise bir görüntüyü oluşturan her bir pikselin komşu pikselleri ile ilişkili yani renk olarak birbirlerine çok yakın olmasıdır. Bir görüntüde yer alan piksellerin bazı tekrarları birleştirilebilir. Sıkıştırma yapılırken orijinal resimden çok uzaklaşmamaya özen gösterilmelidir.[36]

Sıkıştırmanın ana hedefleri dışında bazı alt hedefleri de vardır. Bunlar işlemci ve bellek kullanımının en aza indirilmesidir. Çalışmada bununla ilgili örneklerde verilmiştir. Simülasyon sonuçlarında resim bazında veya algoritma bazında bu detay incelenmiştir.

Görüntü Sıkıştırma işlemi genel hatlarıyla iki sınıfa ayrılır. Bunlar, kayıplı sıkıştırma ve kayıpsız sıkıştırmalardır.

Kayıpsız sıkıştırma modelinde hiçbir bilgi kaybı olmaz istenildiğinde orijinal resme geri dönülebilir. Bu sıkıştırma modeli genel olarak arşivleme ve tıp alanında kullanılır. En popüler, olasılık tabanlı tekniklere dayanan kayıpsız sıkıştırma yöntemi huffman kodlamasıdır[28].

Kayıplı sıkıştırma modelinde ise bilgi kaybı kabul edilebilir. Kayıpsız sıkıştırma yönteminin yetersiz kaldığı veya gerek olmadığı yerlerde kullanılır. İstenilen oranda bilgi kaybı kabul edilerek görünü sıkıştırılır. Kayıplı sıkıştırmaya örnek olarak ise skalar kuantalama yönetimi örnek gösterilebilir.

3.3. K-Means Algoritması

En basit ve anlaşılır yapay zekâ algoritmalarından biri olan k-means, verilen bir veri seti üzerinden belirli sayıda kümeyi (k adet) gruplamak için geliştirilmiş bir algoritmadır. 1967 yılında J.B.MacQueen tarafından geliştirilmiştir [29]. Bu tez içeriğinde bahsedilen veri seti sıkıştırılmak için kullanılan resmin pikselleri olarak düşünülmektedir.

Algoritmanın isminde bulunan K harfi algoritmanın içinde kullandığımız k adet oluşturulan kümeden gelmiştir. K değeri ne ise ortaya çıkan nihai sonuçta da k değeri kadar küme vardır.

K-Means algoritması her noktanın yada verinin sadece bir kümeye ait olması ve her bir merkez noktanın kümeyi temsil etmesi esasına dayanmıştır. Küme içindeki noktaların benzerliği maksimum, küme dışındaki noktaların benzerliği minimum olmalıdır. Buda küme merkezlerini belirlerken seçilen noktaların benzerliğiyle alakalıdır.

Algoritmanın işleyişi şu şekildedir. K sayısı kadar aynı olmayan küme merkezleri belirlenir. Diğer noktaların teker teker her bir küme merkezine olan uzaklığı hesaplanır. Uzaklık hesaplama işlemini genel geçer olarak Öklid Formülü ile bulunur. Fakat Öklid formülünden dışında sıkça kullanılan 2 formül daha vardır. [36]

Bütün iterasyonlarda her bir noktanın her bir kümeye olan uzaklıkları bulunduktan sonra o nokta hangi kümeye en yakınsa o kümeye ait olduğu kabul edilir. Daha sonra her bir kümenin kendi içindeki noktalarının ortalaması alınır ve yeni küme merkezi belirlenmiş olur. Küme merkezleri değişmeyene kadar veya maksimum belirlenen iterasyon sayısına kadar bu işlem devam eder. [37]

Sonuç olarak ise K değişkenine verilen değer kadar küme ve o kümelere ait noktalar kümelenmiş olur.

Uygulanabilirliği çok basit ve büyük veriler için hızlı çalışan bir algoritmadır. Fakat gürültülü verileri ayırmadan bütün noktalar üzerinde işlem yaptığı için gürültüler sonucu saptırabilir.

3.4. KNN Algoritması

KNN Algoritması verilen veri setinin içindeki merkez noktalarda en yakın K sayısı kadar komşusunun değerleri ele alınarak hesaplanır. Burada önemli olan k değeri, uzaklık ölçümü ve verilerin değerleridir [30].

KNN algoritması görüntüdeki noktaları ayırt etmeden gürültüleri ile birlikte hesaplar. Buda bize gürültülü resimlerde güvenilmez bir sonuç verir. K değerinin küçük seçilmesi halinde kümeleme esnasında veri gürültülerden daha çok etkilenir, k değeri

büyük seçilirse daha çok komşu değerlendirmeye katılacağı için biraz daha gürültüsüz bir sonuç elde etmiş olunur [37].

Noktalar arası uzaklıkları hesaplarken genel geçer olarak Öklid fonksiyonu kullanılabilir, noktaların özelliklerine göre bu formül değişiklik gösterebilir. Öklid fonksiyonu yerine Manhattan veya Minkowski fonksiyonu da kullanılabilir [36].

Algoritmanın işleyişi şu şekildedir. Öncelikle K-Means algoritmasına benzer bir şekilde k değeri belirlenir. K değeri olmadan işleme başlanılması mümkün değildir.

Yeni nokta x bir noktada seçilir. Bu noktaya merkez noktada denebilir. Bu noktanın veri setindeki diğer noktalar arasındaki uzaklık hesaplanır. Hesaplanan bu uzaklığa göre küçükten büyüğe doğru sıralanır. Bunun sebebi en yakından komşudan uzaktaki komşuya doğru noktaları sıralamaktır. K değeri kaç olarak belirlendiyse sıralanmış noktaların içindeki ilk k değeri kadar olanlar seçilir. Bu esnada k değerinin tek sayı olmasına dikkat edilmelidir. K değeri kadar olan noktaların içinde verilerin hangisi çoğunluktaysa o kümeye atanır[30].

Eğitim kümesinin büyük olması ve k değerini uygun seçilmesi KNN algoritması açısından çok önemlidir. KNN algoritmasına karar verildiğinde, veri setinin ve k değerini arttırarak sonuçlara bakılır. Sonuç sabitleşmeye başladığında iyi bir KNN algoritması uygulaması yapılmıştır. KNN algoritmasının kötü tarafı her ekleme işleminde uzaklıkların yeniden hesaplanmasıdır[30].

3.5. Delta Sıkıştırma Yöntemi

Delta sıkıştırma algoritması, kayıpsız olarak tabir edilen sıkıştırma yöntemleri arasındadır. Veriyi tam bilgi yerine ardışık veriler arasındaki farklar şeklinde saklar ve karşı tarafa bu farklardan oluşan veri gönderilir. Daha genel anlamda bu veri farklılaştırması olarak bilinir. Adı Delta Encoding olarak da bilinir.[31]

Bu algoritmada verinin boyutunun küçüleceği garanti edilemez. Bunun sebebi veriler arasındaki fark çok büyükse farkını tuttuğumuz sonuç bilgisi de büyük olacaktır. Hatta orijinal sayıdan daha büyük olma ihtimali de vardır.

Algoritma iki aşamadan oluşur. Birinci aşama sıkıştırma aşaması. Bu aşamada ardışık gelen sayıların arasındaki fark kaydedilir ve iletilmesi gereken yere bu şekilde

iletilir. İkinci aşama ise açma aşaması. Bu aşamada ise dizi halindeki farklar sayı kümesi tam tersi bir işlemle açılır. [39]

Sıkıştırma aşamasında ardışık sayılardan ilki direkt olarak kaydedilir. Diğer elemanlar sırayla ilk sayıdan çıkartılarak ilerlenir. Böylece sadece birbirinin farklarından oluşan bir sayı dizisi kalır.

Açma aşamasında ise ilk sayı yine direkt olarak kaydedilir. Diğer elemanlar sırayla ilk sayının üstüne eklenerek ilerlenir. Sonuç olarak kayıpsız olarak orijinal sayı dizisine dönülmüş olur. [39]

3.6. IOS Programlama

IOS Apple şirketinin mobil cihazları için geliştirdiği mobil işletim sistemidir. Çekirdeğini MAC OSX den alır. IOS içinde 4 katman bulunur. Bunlar: Core OS tabakası, Core Servisleri tabakası, Medya tabakası ve CocoaTouch tabakasıdır [32].

IOS programlamak için iki dil seçeneği vardır. Apple ilk olarak Objective-C ile başlamış daha sonra Swift adlı kendi ürettiği yazılım diliyle devam etmiştir.

Objective-C; dil olarak C tabanlıdır. Fakat C'den oldukça farklı özelliklere sahiptir. Nesne yönelimli bir programlama dilidir. 1980'lerde Brad Cox tarafından Stepstone adlı şirkette geliştirilmiştir.

Swift; Apple tarafından kendi cihazları için program geliştirebilmek amacıyla yazılan bir dildir. Piyasaya ilk çıkışı 2014 yılındadır. Objective-C'ye göre daha kolay ve basit bir yapıya sahiptir.

İki dilinde geliştirme ortamı X-Code programıdır. X-code sadece MacOS işletim sisteminde çalışan bir programdır. Yani Mac bilgisayar olmadan gerçek anlamda IOS programlama yapılamamaktadır. Virtual Machine üzerinde de geliştirme yapılabilir fakat simülatörün hızı ve stabilite mac bilgisayar üstünde daha fazladır.

Program yazıldıktan sonra kullanıcılara ulaşmak için Apple şirketinin uygulamalarını kütüphane olarak tuttuğu AppStore kütüphanesine yüklemek gerekir. Bunun için ise bir developer hesabı oluşturmak ve bu kimlikle kütüphaneye uygulamayı yüklemek gerekmektedir. Bu sayede kullanıcılar uygulamayı direk indirip kullanabilmektedir. Başka bir yöntem ise geçici olarak debug amacı ile geliştirme ortamında direkt olarak kablo aracılığı ile telefonlara veya cihazlara yüklemektir.

3.7. Socket.IO Programlama

Her geçen gün internet kullanımı bir önceki güne göre sürekli bir artış içindedir. Yapılan uygulama veya yazılımları kullanan kişi sayıları binlerle, milyonlarla ifade edilebilir. Bu denli anlık trafiği olan yazılımlarda ölçeklenebilirlik çok büyük sorun teşkil etmektedir. Özellikle eş zamanlama başlı başına bir problemdir. Normal sistemlerde bu denli kalabalık trafiği olan yazılımların ölçeklenebilirliği çok yüksek maliyetlere ihtiyaç duyar. Node.JS bu tür sorunları çözmek için ortaya çıkan bir çözüm ürünüdür.

Node.JS için kısaca javascript dilinin sunucu taraflı olanı denilebilir. Joyent tarafından 2009 yılında ortaya çıkarılmış bir kütüphanedir. V8 adı verilen javascript motoru üzerinde çalışır. V8 ise Google tarafından geliştirilen açık kaynaklı bir javascript motorudur[33]. Aslen V8,Google Chrome tarayıcısı için geliştirilmiştir fakat Node.JS, MongoDB gibi projelerde kullanılmaktadır. Node.JS'in amacı javascript kodlarını makine koduna çevirerek sunucu tarafında non-blocking olarak asenkron bir biçimde işlemleri yapmaktır. Socket.IO ise Node.JS üzerine kurulan bir kütüphanedir.

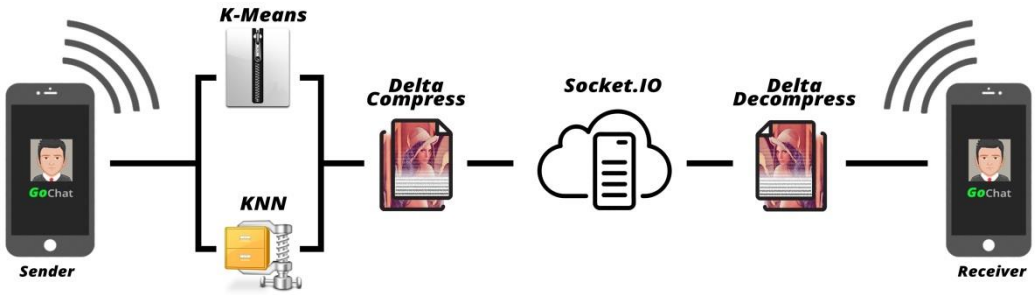
4. GELİŞTİRİLEN SİSTEM

Bu bölümde tez içeriğinde geliştirilen sistemin genel bilgileriyle birlikte detayları açıklanmaktadır. Öncelikle ortaya çıkan uygulamanın ana fikri anlatılmaktadır. Devamında ise sözde kod, akış şeması ve uygulamanın arayüzleri tanıtılmıştır.

4.1. Genel Fikir

Bu çalışmada IOS işletim sistemli cihazlar için Socket.io tabanlı olarak mesajlaşma ve aynı zamanda yapay zeka algoritmaları kullanılarak görüntü sıkıştırma ve gönderme uygulaması geliştirilmiştir. Sistem tamamen IOS işletim sistemi ile çalışan mobil cihazlarda çalışmaktadır. Socket.io kullanılarak gerçek zamanlı mesajlaşma sağlanmıştır.

Sohbet esnasında metin mesajı dışında görüntü göndermek istenildiğinde Şekil 4.1'de görüldüğü gibi bu görüntüyü seçilen iki yapay zeka algoritmasından biri ile sıkıştırmaktadır. Sıkıştırılan görüntüyü Delta Sıkıştırma algoritması ile string'e çevirerek socket.io aracılığı ile diğer kullanıcılara gönderir. Diğer cihazlarda ise Delta Sıkıştırma algoritmasını tersine çevirerek gelen string görüntüyü resim olarak kullanıcıya gösterir.



Şekil 4.1: Geliştirilen sistemin özet akış şeması.

Yapay zeka algoritması olarak K-Means ve KNN algoritmaları kullanıcıya seçenekli olarak sunulmuştur. Kullanıcı istediği algoritmayla resmi sıkıştırabilir. Bunun amacı kullanıcının seçiminden ziyade iki algoritma arasında ki resim bazında performans farkını görebilmektir. Resimler sıkıştırılarak gönderildiğinde ortaya çıkan enerji kullanımı, cihazın ram kullanımı gibi kaynak kullanımların analizi yapılmaktadır.

4.2. Sözde Kod

Uygulamanın işleyişini anlatan kod Şekil 4.2'de gösterilmiştir. Kullanıcının hangi adımları uygulayacağı ve hangi kodları çalıştıracağı ana mantık olarak anlatılmıştır.

```
Geliştirilen Sistem
Uygulamayı aç.
kullaniciAd= Kullanıcı adı gir.
İf(kullaiciAd==’’)
        Goto “kullaniciAd= Kullanıcı adı gir.“
Endif
Sunucuda kayıtlı kullanıcı listesini getir.
Joinchat butonuna tıkla.
Mesaj listesini getir.

mesaj yaz ve send butonuna tıkla
İf(mesaj!=””)
        Mesajı gönder.
Endif

Image butonuna tıkla.
Choose Image butonuna tıkla ve Resim seç.
Cluster Count belirle.
İf(KMeans)
        KMeans ile sıkıştır.
        Send butonuna tıkla.
        Görüntüyü gönder
Else if(KNN)
        Knn ile Sıkıştır.
        Send butonuna tıkla.
        Görüntüyü gönder
Endif
```

Şekil 4.2: Geliştirilen sistemin sözde kodu.

Uygulamada yapay zeka alt yapısı olarak kullanılan K-Means algoritmasının resim ile beraber çalışması Şekil 4.3'te sözde kod olarak anlatılmıştır.

K-Means Algoritması

P: Seçilen resmin her bir noktasını piksel değerleriyle beraber bir diziye ata.
K: Bu dizinin içinden k değeri kadar birbirinden faklı nokta seç ve diziye ata

```
While K=PreviousK
    For i=0;i<p.count;i++
        For j=0;j<K.count;j++
            Pnin içindeki i indeksli noktanın Knın içindeki k indeksli
    noktaya uzaklığı Öklid Hesaplaması ile bulunsun
        Endfor j
        P'nin içindeki i indeksli nokta for j'deki hangi noktaya en yakınsa o
    kümeye atansın
    Endfor i
    Her K dizisine atanan P noktalarının ortalamaları alınır.
    Yeni K noktası belirlenir.
EndWhile
```

Şekil 4.3: Uygulanan K-Means algoritmasının sözde kodu.

Uygulamada yapay zeka alt yapısı olarak kullanılan bir diğer algoritma olan KNN algoritmasının resim ile beraber çalışması Şekil 4.4'te sözde kod olarak anlatılmıştır.

P: Seçilen resmin her bir noktasını piksel değerleriyle beraber bir diziye ata.

K: Bu dizinin içinden k değeri kadar birbirinden faklı nokta seç ve diziye ata

For i=0;i<p.count;i++

For j=0;j<K.count;j++

Pnin içindeki i indeksli noktanın Knın içindeki k indeksli noktaya uzaklığı Öklid Hesaplaması ile bulunsun

Endfor j

P'nin içindeki i indeksli nokta for j'deki hangi noktaya en yakınsa o kümeye atansın

Endfor i

Her K dizisine atanan P noktalarının sıralaması en yakından uzağa sıralanır.

İlk sıradan başlayıp seçilen k değeri kadar olan noktaların içinde hangi renk fazlaysa o renk yeni merkez kümesi olur.

Şekil 4.4: Uygulanan KNN algoritmasının sözde kodu

Uygulamada görüntü sıkıştırma başlığı altında alt yapı olarak kullanılan bir diğer algoritma olan Delta algoritmasının resim ile beraber çalışması Şekil 4.5'te anlatılmıştır.

Delta Algoritması

P: Seçilen resmin her bir noktasını piksel değerleriyle beraber bir diziye ata.

Sonuc: boş degisken oluştur.

Sonuç+= p[0] noktaların ilk değerini sabit olarak olduğu gibi atıyoruz.

For i=1;i<p.count;i++

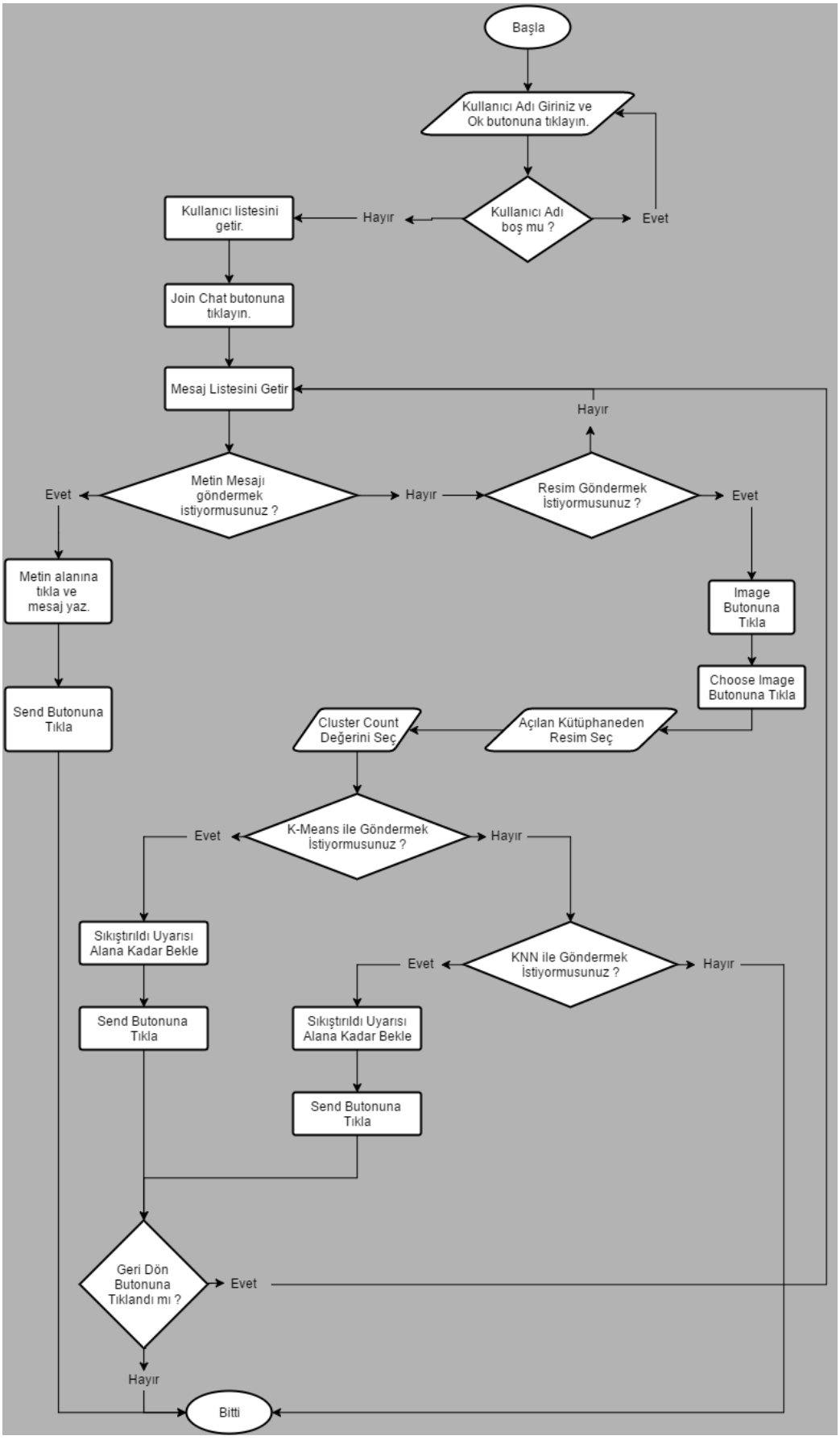
Sonuç += p[i]-p[i-1] döngüde şu anki indeksinin değeri ile bir önceki indeksinin değeri çıkartılıp sonuca eklenir.

Endfor i

Şekil 4.5: Uygulamada resmi göndermek için sıkıştıran Delta Algoritması

4.3. Akış Şeması

Geliştirilen sistemin akış diyagramı Şekil 4.6'da gösterilmektedir. Sistemin kullanıcı tarafında işleyişi akış diyagramından açıkça takip edilebilmektedir.

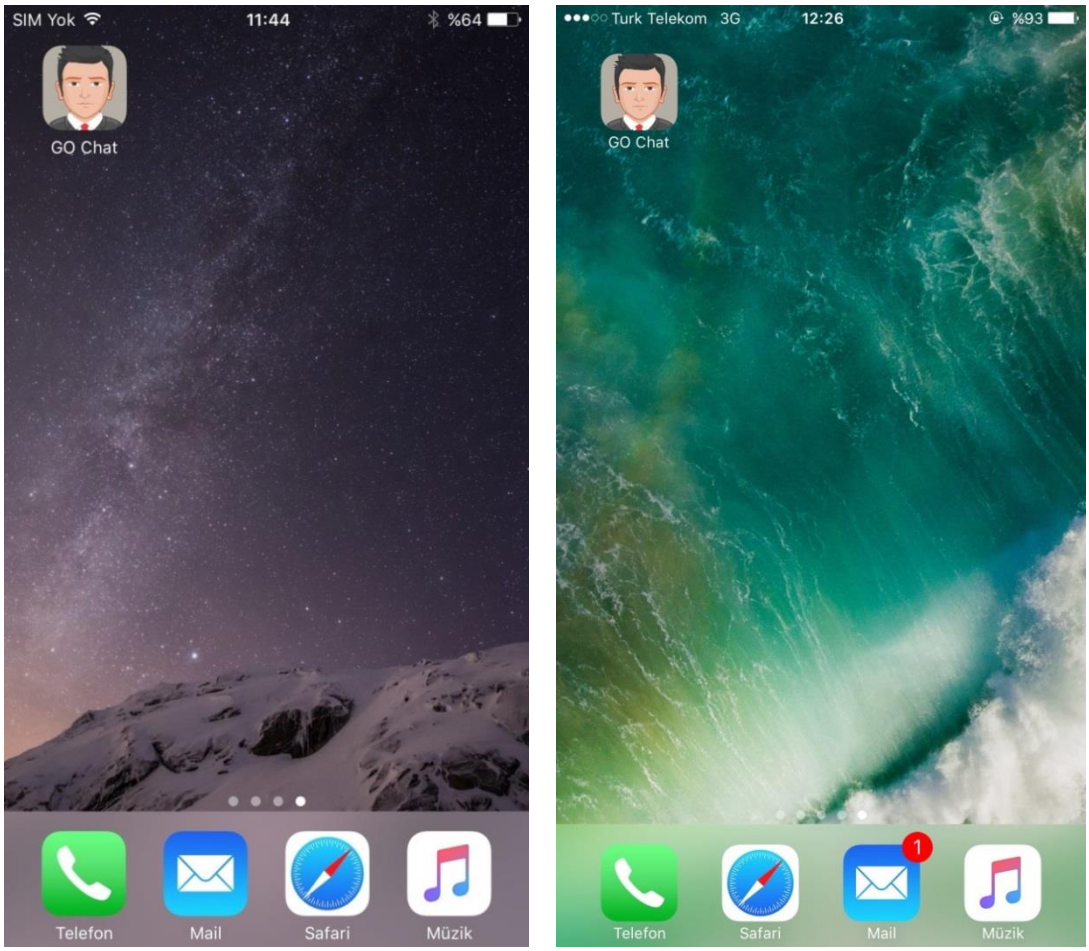


Şekil 4.6: Geliştirilen sisteme ait akış diyagramı.

4.4. Ara Yüzler

Uygulamayı aynı anda kullanan iki cihazdan alınan ekran görüntüleri ve fotoğraflar aşağıda gösterilmektedir. Kullanılan cihazların modeli IPhone 6 Plus modelidir. Fakat uygulamanın normalde herhangi bir model kısıtı yoktur. IOS 10 ve üzeri her cihazda çalışır.

Uygulama Apple şirketinin uygulama dükkânı olarak da bilinen AppStore'dan indirildikten sonra veya debug modda telefona yüklendikten sonra, uygulamanın ikonu ve görüntüsünün bulunduğu ekran Şekil4.7'de görüldüğü gibidir. Bu ikona tıklayıp uygulama başlatılabilir.



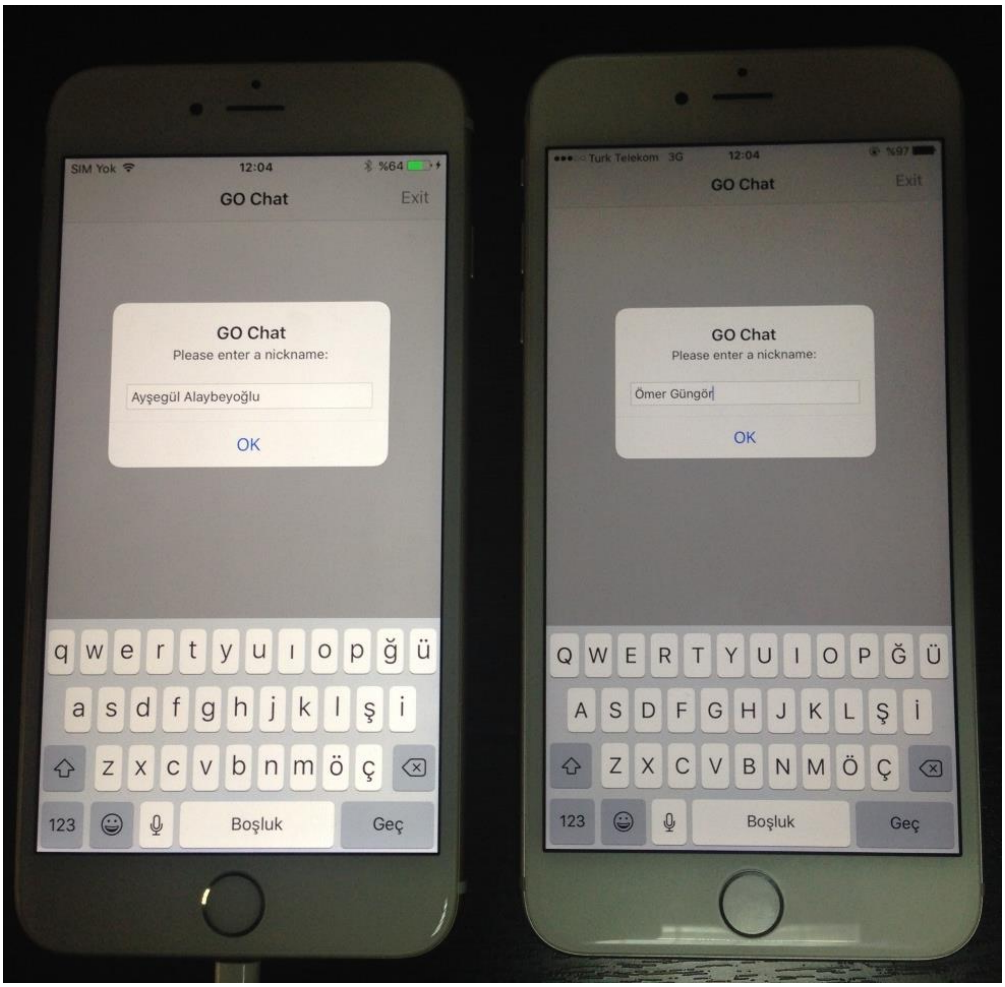
a-)Gönderici **b-)Alıcı**

Şekil 4.7: Uygulamanın ana ekran görüntüsü.

Uygulama ikonu cihazın herhangi bir sayfasında olabilir. İlk yüklendiğinde boş bulduğu ilk sayfaya otomatik yerleştirilir. Daha sonrasında kullanıcı uygulamayı istediği yere taşıyabilir. Uygulamanın ismi ve ikonu değişemez. Yeni bir yükleme

veya güncelleme yapılmadığı sürece kullanıcı tarafından değişimi imkânsızdır.

Uygulamada sohbet odasına katılabilmek için kullanıcıların birer ismi olması gerekmektedir. Uygulama açıldığında bunun için Şekil 4.8'de görüldüğü gibi bir kullanıcı adı istenir. Gönderilen iletilerde veya resimlerde hep bu isim gözükmektedir. Kullanıcı isim girip tamama tıklandıktan sonra kullanıcı sohbet odasına dâhil edilir.

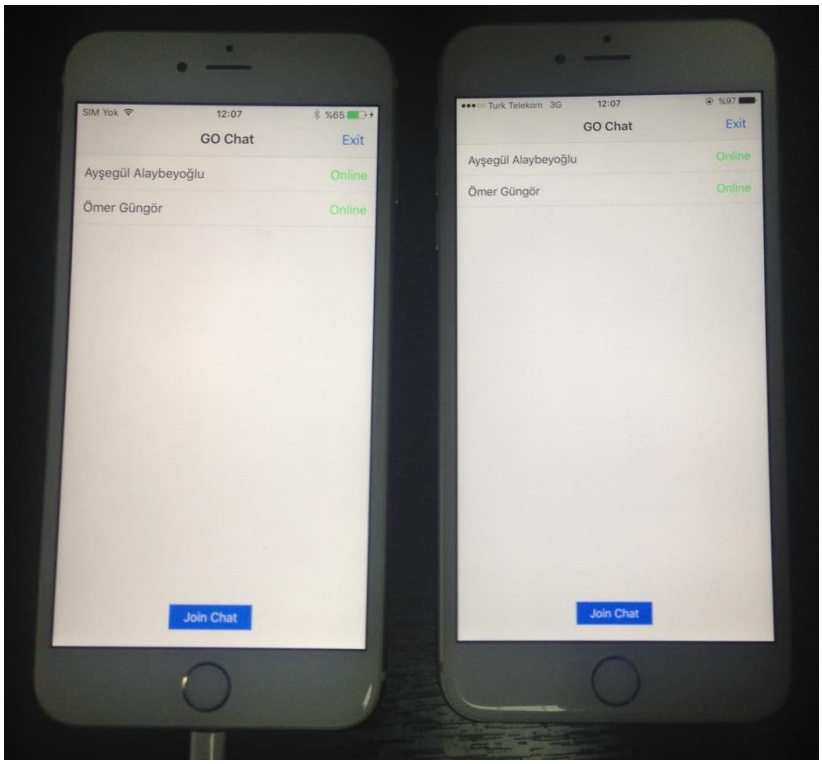


Şekil 4.8: Uygulamada sohbete katılabilmek için isim girme bölümü.

İsim girilmeden bu ekran geçilemez. Uygulamadan çıkılması veya isim girilmesi gerekir. Bunun sebebi, uygulama içinde her şey uygulamayı kullanan, sohbet eden kullanıcının kullanıcı adı üzerinden ilerlemektedir.

Uygulamaya daha sonra tekrar girildiğinde aynı kullanıcı isimle kullanılırsa yine aynı kullanıcı üzerinden devam edilmiş olur. Aksi takdirde sunucuda yeni bir kullanıcı oluşturulacaktır.

Şekil 4.9'da ki görüntü kullanıcı adı girdikten sonra açılan ekrandır. Bu ekranda kullanıcı adı girip sunucuya bağlanmış olan kullanıcıların listesi görülür. Şekil 4.9'da iki kullanıcıda da görüldüğü gibi birbirlerini Online olarak görmüşlerdir. Uygulamadan çıkıldığında ise çıkan kişinin yanında Online yerine Offline ibaresi yer alır. Aynı isimde başka bir kullanıcı giriş yapmadığı sürece listedeki o isim Offline olarak kalacaktır.

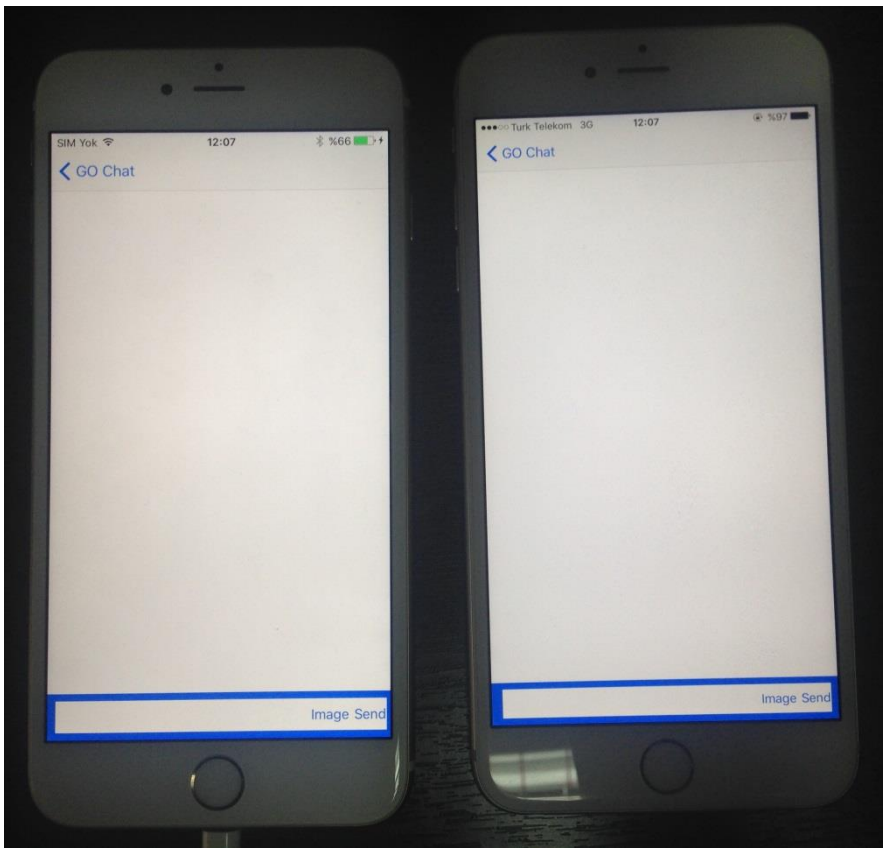


Şekil 4.9: Uygulamanın sohbette olan kişi listesi ve listedeki kişilerin çevrim durumları.

Bir kullanıcı farklı zamanlarda aynı cihazdan değişik isimlerle uygulamaya giriş yapabilir. Bu işlem herhangi bir sorun teşkil etmez. Kullanıcı listesinde oluşan atıl kayıt dışında, yapılan işlemin sakıncası yoktur. Sunucu yeniden başlatılana kadar listede o kullanıcı ismi hep görünür.

Şekil 4.10'da ki görüntü JoinChat butonuna tıkladığımızda karşımıza gelen ekrandır. Bu ekran kullanıcıların sohbet edebilecekleri veya resimleri görebilecekleri ekrandır.

Anlık olarak bu ekranda kalındığı sürece gönderilen her şeyi burada görülebildiği gibi ekranın altındaki yazı yazılabilen alana tıklanıp, ileti yazıp, Send butonuna ile gönderilebilir. Ayrıca araştırmanın amacı olarak görünen resim işlemlerinde Image butonu ile yapılabilir.



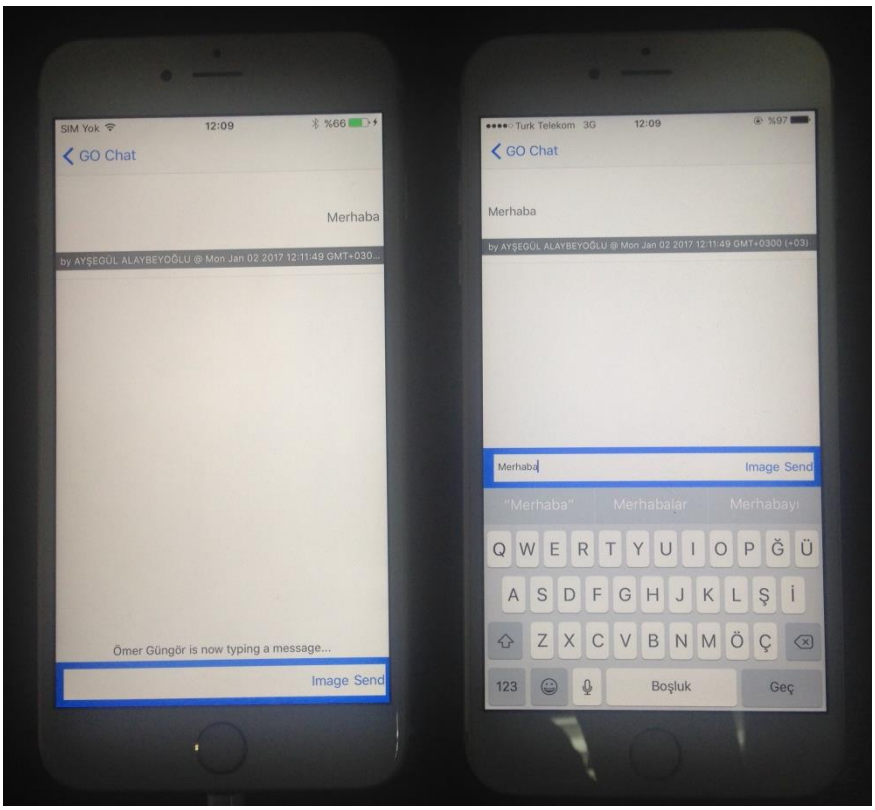
Şekil 4.10: Gönderici uygulamada kullanıcı sohbet etmek için odaya ilk girdiğinde görünen ekran görüntüsü.

Gönderilen resimler Şekil 4.10'da görünen ekranda, tablo her yüklendiğinde delta algoritması tarafından decompress edilir. Bunun dezavantajı ekranda çok resim olduğunda resim sayısıyla doğru orantılı olarak bekleme süresi oluşmasıdır.

Sohbet ekranındayken giriş yapılan kullanıcı dışında herhangi bir kullanıcı ileti yazmaya başladığında, Şekil 4.11'de olduğu gibi ileti yazan kullanıcının kullanıcı adıyla birlikte sunucuya bildirilir. Sunucuda diğer kullanıcılara bu durumu bildirir.

Bu durumun sohbet esnasında karışık konuşmaları bir oranda azaltmaktadır. Karşıdaki kullanıcının mesaj yazdığını gören kullanıcı mesaj yazmayı bıraktığında daha okunabilir ve anlaşılabilir sohbet odası oluşacaktır.

Yazı yazma işlemi bırakıldığında eğer yazı alanında herhangi bir harf veya karakter yoksa bildirim iptal edilir. Aynı durum sohbet ekranından çıkıldığında da aynıdır. Yazı yazan bir kullanıcı sohbet ekranından çıktığında diğer kullanıcılardaki mesaj yazıyor bildirimi kaybolacaktır.



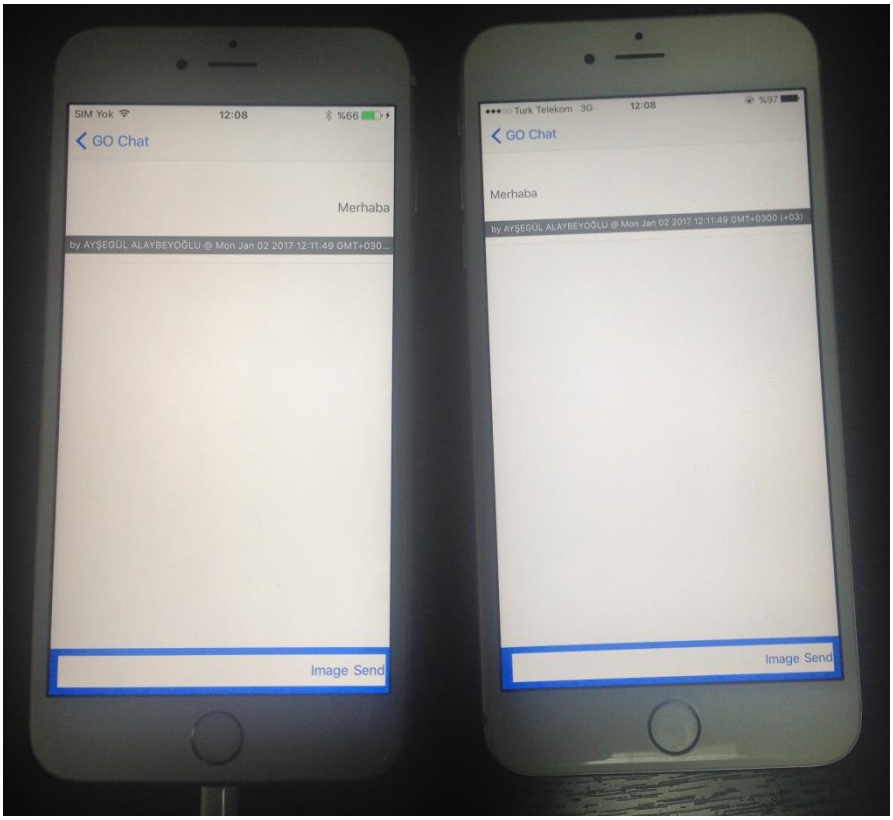
Şekil 4.11: Diğer kullanıcılar tarafından mesaj yazılmaya başlandığında gelen bildirim.

Giriş yapılan kullanıcı dışında herhangi bir kullanıcı ileti gönderdiğinde Şekil 4.12'deki gibi bir ekran görüntüsü oluşur. Giriş yapılan kullanıcının mesajları dışındaki tüm mesajlar sola dayalı şekilde görünür.

Ayrıca gönderilen mesajın alt kısmında koyu renkli bir bant vardır. Bulunduğu mesaj ile ilgili bilgileri içerir. Kim tarafından gönderildiği, ne zaman gönderildiği bilgi amacıyla bu bandın içinde görünür.

Şekil 4.12'deki ekranda kullanıcı sürekli sunucuyu dinleme modunda bekliyor olacaktır. Diğer kullanıcılardan yeni mesaj veya görüntü olmadığı sürece bu ekranda bir hareket görülemez.

Diğer kullanıcılar tarafından gönderilen metin mesajları ve görüntüler gönderilme zamanına göre ilk gönderilen yukarıda olacak şekilde listelenir.

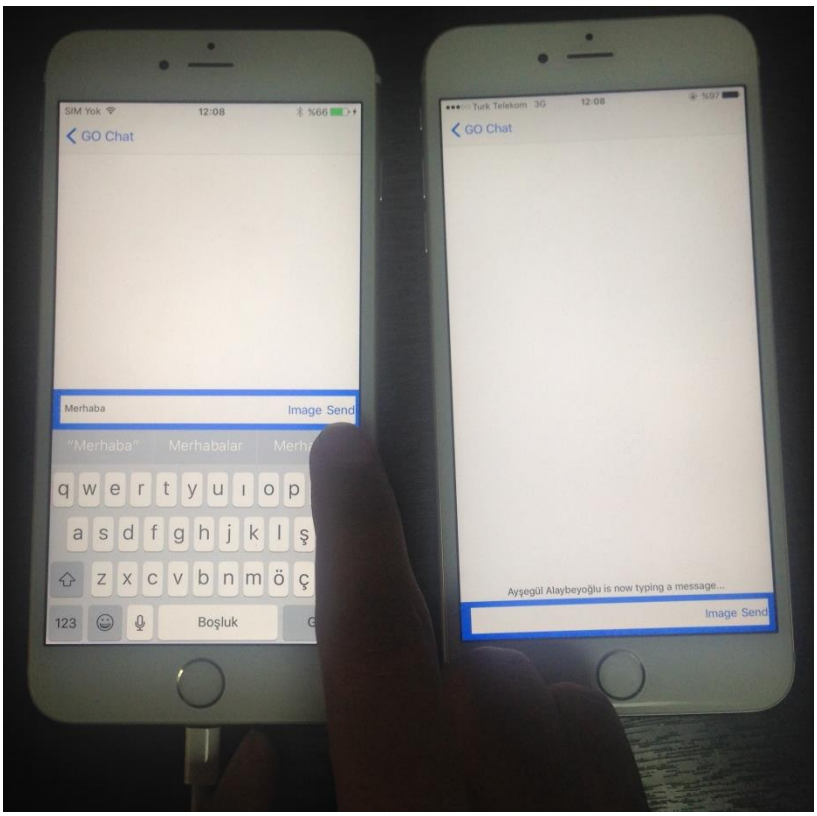


Şekil 4.12: Sohbet ekranında bir kullanıcı tarafından gönderilen metin mesajı.

Sohbet ekranında alt taraftaki metin mesajı yazılabilen bölüme tıklandığında Şekil 4.13'de görüldüğü üzere klavye açılır. Açılan bu klavyeden istenilen uzunlukta metin mesajı gönderilebilir.

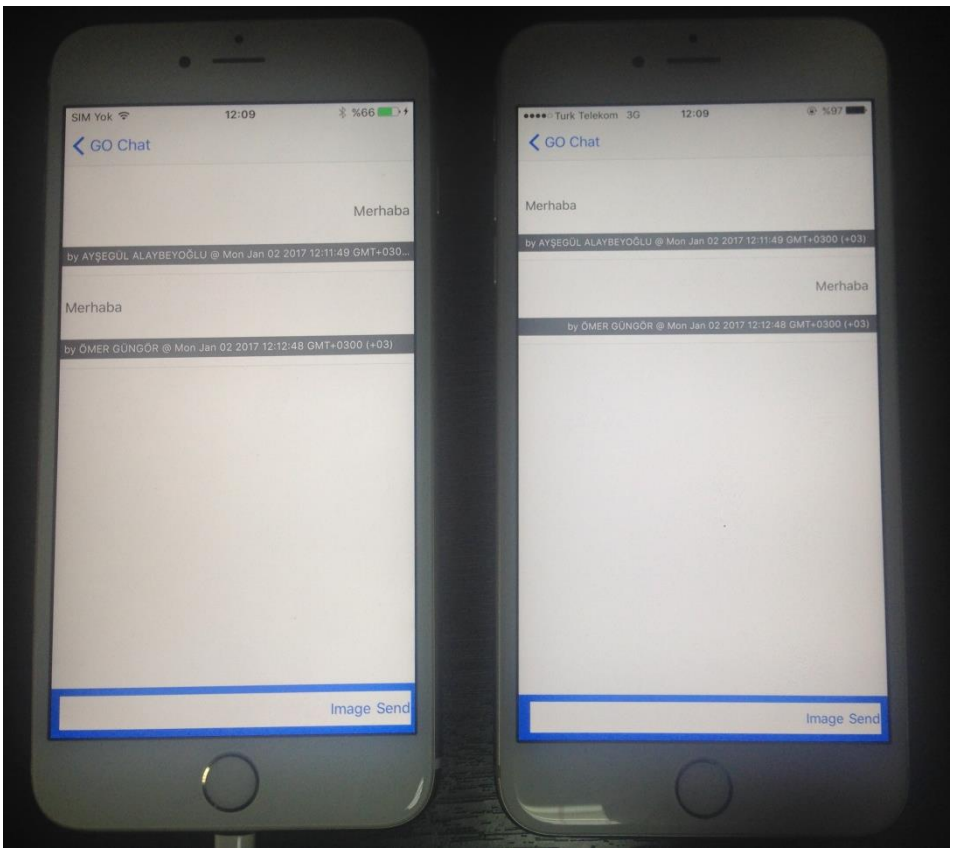
Metin mesajını yazdıktan sonra Send butonu ile önce sunucuya oradan da sunucudaki yeni mesajları dinleyen tüm sohbet odasındaki diğer kullanıcılara iletmiş oluyoruz. Metin mesajı yazdığımız esnada diğer kullanıcılarda, giriş yapılan kullanıcı adıyla, mesaj yazdığımıza dair bildirim gitmiş oluyor.

Mesaj gönderildiğinde o anda sohbet odasında olmayan kullanıcılar mesajı bir daha geriye dönük olarak göremezler veya kullanıcılara yeni mesaj olduğuna dair bir mesaj gitmez. Kullanıcılar sadece sohbet ekranında dinleme modunda iken metin mesajı veya görüntü alabilirler.



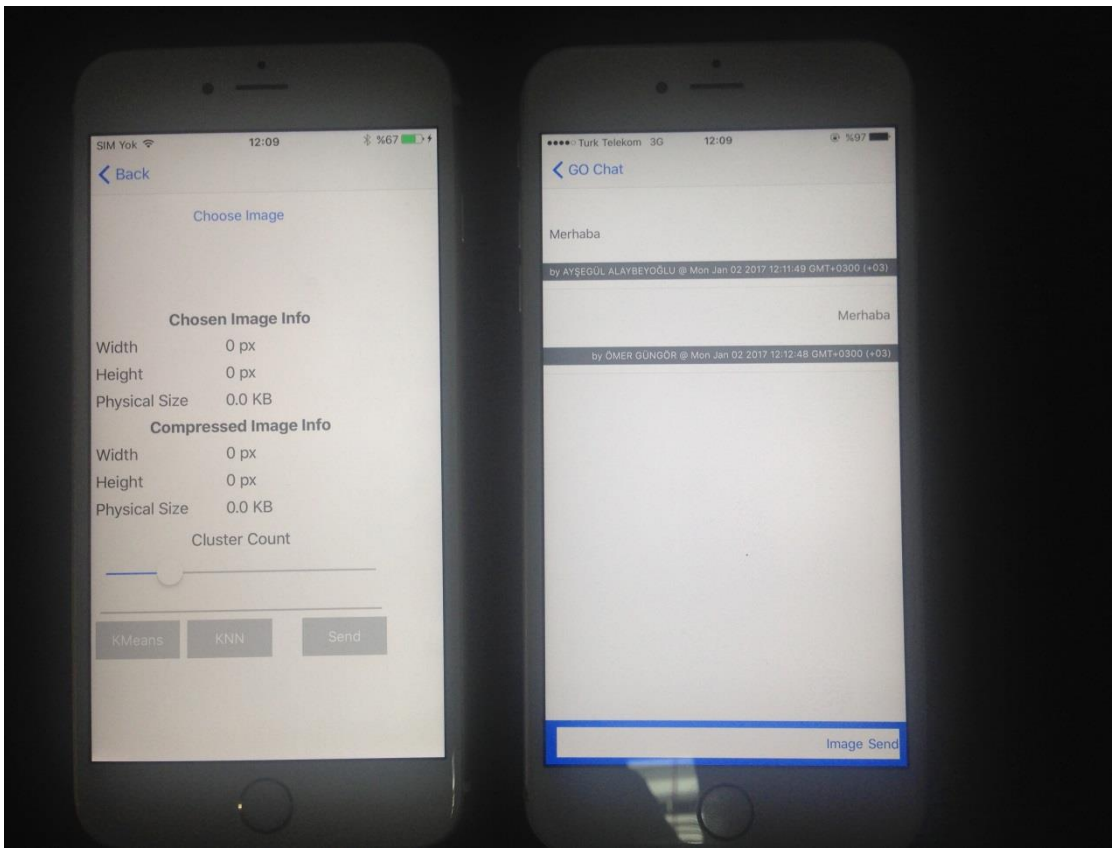
Şekil 4.13: Sohbet ekranında metin mesajı gönderirken görünen ekran görüntüsü.

Metin mesajı yazıp Send butonuna tıklandığında ekranda oluşacak görüntü Şekil 4.14'deki gibidir. Genel olarak yapılan mesajlaşmalarda bu şekilde bir görüntü olacaktır. Şekil 10'daki gibi her kullanıcıdan bir tane değil, herhangi bir kullanıcı istediği kadar mesajı arka arkaya yazabilir, gönderebilir. Aynı şekilde diğer kullanıcıların arka arkaya yazdığı mesajları da görebilir.



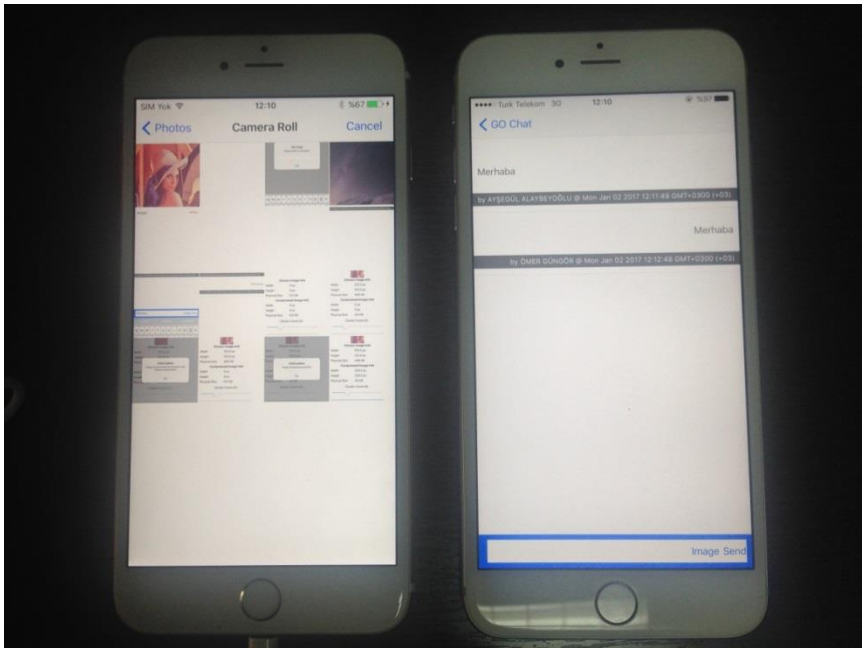
Şekil 4.14: Sohbet ekranında kullanıcı metin mesajı gönderdiğinde ekrana gelen görüntü.

Image butonuna tıklandığında Şekil 4.15'de ki gibi bir görüntü ekrana gelir. Bu ekranda resim seçip istenilen algoritmaya göre sıkıştırılıp, orijinal resim ve sıkıştırılmış resim bilgilerini görüp istediğimiz zamanda gönderme işlemi yapabiliriz.



Şekil 4.15: Image butonuna tıklandığında görünen ekran.

Choose Image butonuna tıklandığında ekrana Şekil 4.16'da olduğu gibi IOS cihazın medya kütüphanesi gelir. Bu ekranda istediğimiz klasördeki resimlerden bir tane seçtiğimizde ekranı otomatik kapatacaktır.

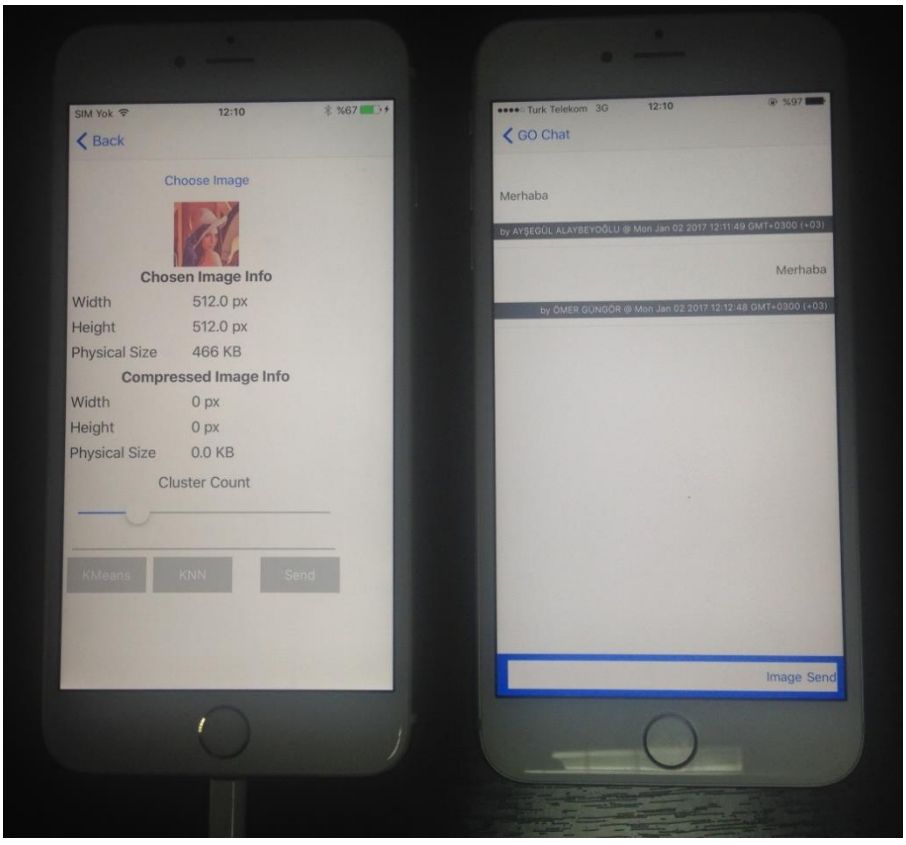


Şekil 4.16: Choose Image butonuna tıklandığında görünen ekran.

Uygulamanın içinde kod ile kütüphanedeki medya türlerinden hangisinin seçilmesi gerektiği belirlenebilir. Bu sayede kullanıcının yanlış veri tipi seçmesi engellenmiş olur. Tezde bahsi geçen uygulamada kullanıcıların sadece fotoğraf seçmelerine izin verilmiştir.

IOS Uygulamalarda bu ekrana kod tarafından ulaşabilmek için cihaz sahibinin izni olması gerekmektedir. Kullanıcı uygulamanın fotoğraflara ulaşmasını istemediği durumlarda, uygulama işlem yapamaz. İlk kullanımda bu izin kullanıcı tarafına sorulur. Kullanıcının verdiği cevap cihaz tarafından tutulur ve bir dahaki seferlerde tekrar tekrar kullanıcıya uygulamanın fotoğraflara ulaşmak istediği uyarısı verilmez. Bu kontrol sayesinde kötü niyetli yazılımların, bu cihazları kullanan kullanıcıların özel bilgilerine ulaşması engellenmiş olur.

Kitaplıktan Resim seçildiğinde otomatik resim seçme ekranı kapanır ve Şekil 4.17'teki ekran gelir. Burada seçilen görüntünün genişlik ve yükseklik bilgilerini görebiliriz. Ayrıca sıkıştırma sonrası karşılaştırmak için Fiziksel boyutu da görebiliriz. Bu bilgiler otomatik değil uygulamayı geliştiren kişi veya kişiler tarafından yazılır.

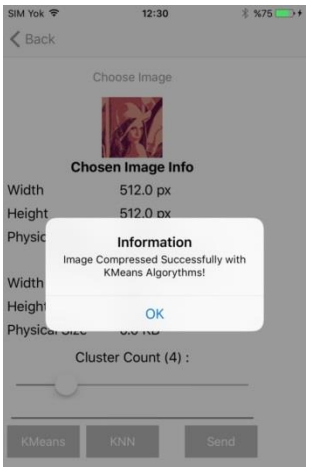


Şekil 4.17: Kitaplıklardan resim seçildiğinde gelen ekran.

Şekil 4.18'de ki ekranda yer alan slider elementi ile resmin bir nevi sıkıştırma oranı olan Cluster Count değerini seçebiliriz. Bu değer K-Means için kaç adet küme oluşturacağımızı, KNN içinse uzaklıkları küçükten büyüğe sıraladıktan sonra kaç küçükten büyüğe doğru kaç adet değeri örnek alacağımızı gösterir.

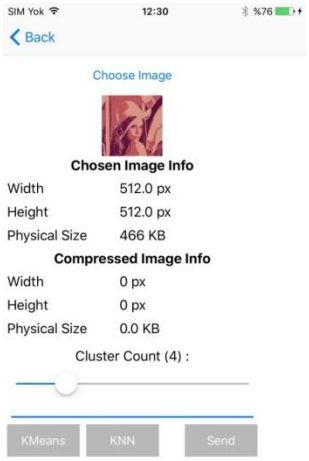
Şekil 4.18: Cluster Count Seçimi.

Cluster Count seçildikten sonra K-Means butonuna tıklandığında seçilen resmi alıp sıkıştırma işlemine başlayacaktır. Bu işlemin süresi Cluster Count değerine göre farklılık gösterecektir. İşlem bittikten sonra ise bize Şekil 4.19'da olduğu gibi "Image Compressed Successfully with Kmeans Algorythms" bilgisi verecektir.



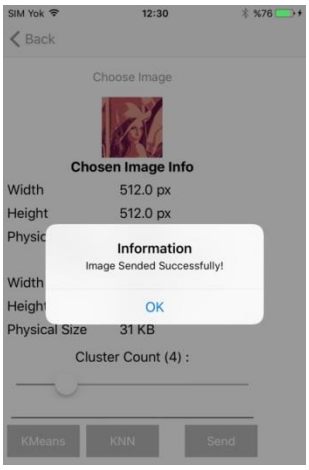
Şekil 4.19: K-Means algoritması ile sıkıştırma uyarısı.

Sıkıştırma işleminden sonra seçilen resmin sıkıştırılmış hali Şekil 4.20'deki gibi ekranda görünmektedir. Bu aşamadan sonra artık tek yapılması gereken Send butonu ile sıkıştırılmış halini göndermektir.



Şekil 4.20: Sıkıştırmadan sonraki görüntü.

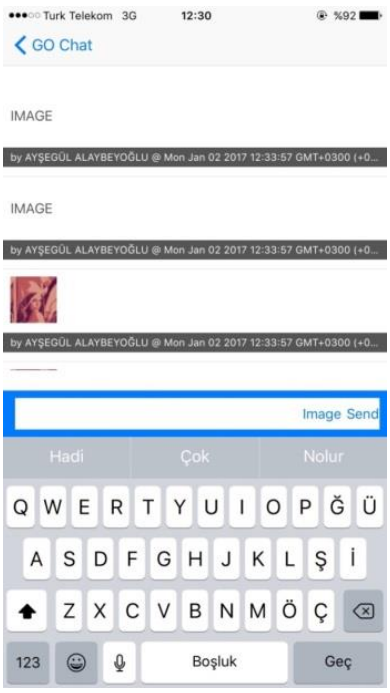
Send butonuna tıklandıktan sonra sıkıştırılan resmi alıp delta algoritmasıyla string değere çevirip karşıya gönderir ve kullanıcıya Şekil 4.21'de olduğu gibi "Image Sended Successfully" mesajını verir. Ayrıca ekrana sıkıştırılmış resmin genişlik yükseklik ve fiziksel boyutunu yazar.



Şekil 4.21: Send Butonuna tıklandığında ekrana çıkan uyarı.

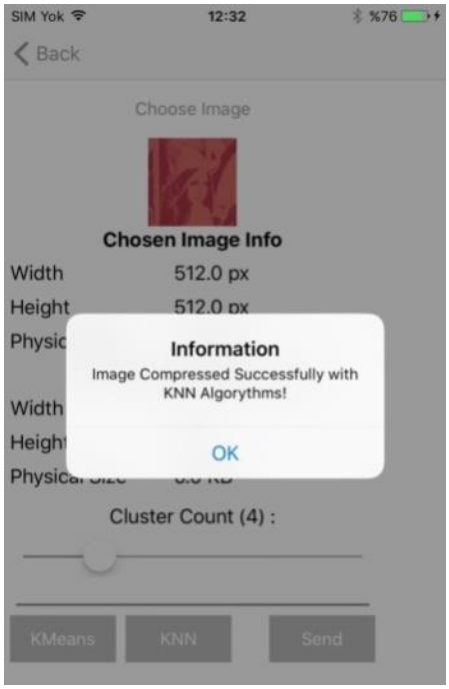
Resmi gönderdikten sonra sohbet odasına dönüldüğünde gönderilen resim gönderici ekranında Şekil 4.22'deki gibi, alıcı ekranında ise Şekil 4.23'deki gibi görünmektedir. Giriş yapan kullanıcı gönderdiyse sağda diğer kullanıcılar gönderdiyse solda görünür.

Şekil 4.22: Gönderici resim gönderdikten sonra sohbet odası.



Şekil 4.23: Alıcı resim aldıktan sonra sohbet odası.

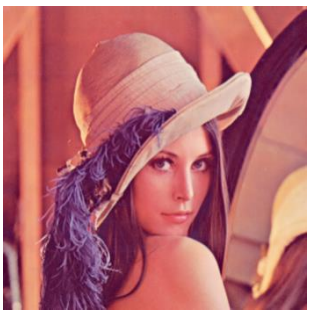
KNN butonuna tıklandığında seçilen resmi algoritmaya göre sıkıştırır ve ekrana Şekil 4.24'de olduğu gibi "Image Compressed Successfully with KNN Algorythms" mesajı gelir. Resim sıkıştırılmıştır fakat bu aşamada henüz gönderilmemiştir. OK butonuna tıklandıktan sonra Send butonuna tıklandığında gönderilecektir.



Şekil 4.24: KNN algoritması ile sıkıştırma uyarısı.

5. SİMÜLASYON SONUÇLARI

Uygulamayı test sürecinde 4 resim kullanılmıştır. Seçilen görüntüler görüntü işleme ile ilgili standart test resimlerinden bazılarıdır. Bu resimler Şekil 5.1.a'da Lena, Şekil 5.1.b'de Pool, Şekil 5.1.c'de ki Cat ve Şekil 5.1.d'de ki Barbara dosyalarıdır. Şekil 5.1.a'daki Lena resmi 500 kilobyte boyutunda olup, 512 piksel x 512 piksel çözünürlüğündedir. Şekil 5.1.b'deki Pool resmi 182 kilobyte olup, 510 piksel x 383 piksel çözünürlüğündedir. Şekil 5.1.c'deki Cat resmi 647 kilobyte olup, 490 piksel x 733 piksel çözünürlüğündedir. Şekil 5.1.d'deki Barbara resmi 181 kilobyte olup, 512 piksel x 512 piksel çözünürlüğündedir.



a–) Lena

b–)Pool

c–)Cat

d–)Barbara

Şekil 5.1: Testlerde kullanılacak deneme resimleri.

Test edilen kriterler enerji kriteri, sıkıştırma oranı yani fiziksel boyut kriteri, ram kullanım kriteri ve zaman kriteridir. Bu 4 kritere göre dört resim K-Means ve KNN algoritmaları tarafından ayrı ayrı test edilip aşağıda sonuçları yazılmıştır.

IOS cihazlarda yapılan uygulamanın testini yapabilmek için xcode derleyicisinin kendine ait olan araçları kullanılır. Apple dışarıya kapalı bir firma olarak bilindiği için bu konuda da kendi test aracı kullanılması zorunludur. İki adet test aracı vardır. Bir tanesi debug durumundayken kullanılabilen Debug Gauges aracı. Diğeri ise Instruments adı verilen xcode derleyicisinin içinde fakat bağımsız çalışan bir araçtır.

Debug Gauges yetenekleri CPU Report, Memory Report, EnergyImpact, Disk Report, Network Report olarak sıralanabilir. Debug Gauges, uygulamanızın sistem kaynakları kullanımında nasıl performans gösterdiğine ilişkin bilgiler sağlar. Uygulamanın özelliklerine ve hedef özelliklerine bağlı olarak uygulamanızın sistemdeki ve diğer çalışan uygulamalar üzerindeki etkisini rapor edebilir. Bu teze konu olan uygulama test edilirken Memory Report ve EnergyImpact yetenekleri kullanılmıştır.

Debug Gauges aracının EnergyImpact raporu, uygulama kullanıldığı esnada kullanılan enerji ile ilgili bazı bilgiler verir. Bu bilgiler şunlardır;

UtilizationEnergyImpact: Anlık olarak yapılan işlemin etkisini gösterir. Zero, Low, High, Very High değerleri alabilir.

AverageEnergyImpact: Debug işlemi başladığından itibaren tüm enerji etkilerinin yüzde olarak ortalamasını verir.

AverageOverhead: Bu işi yapmak için gerekli olan radyo ve diğer sistem kaynaklarını yetiştirmenin bir sonucu olarak enerji kullanımını temsil eder.

Yapılan testlerde iki adet Apple IPhone 6 Plus cihazı ve IOS 10.2 İşletim Sistemi kullanılmıştır. Yapılacak olan testlerde ortaya çıkacak grafikler cihazın modeline, işletim sistemine, aynı anda başka uygulamanın çalışıp çalışmadığına, cihazın bataryasının ömrüne ve cihazın şarjda olup olmamasına ve seçilen resme göre değişiklik gösterebilir.

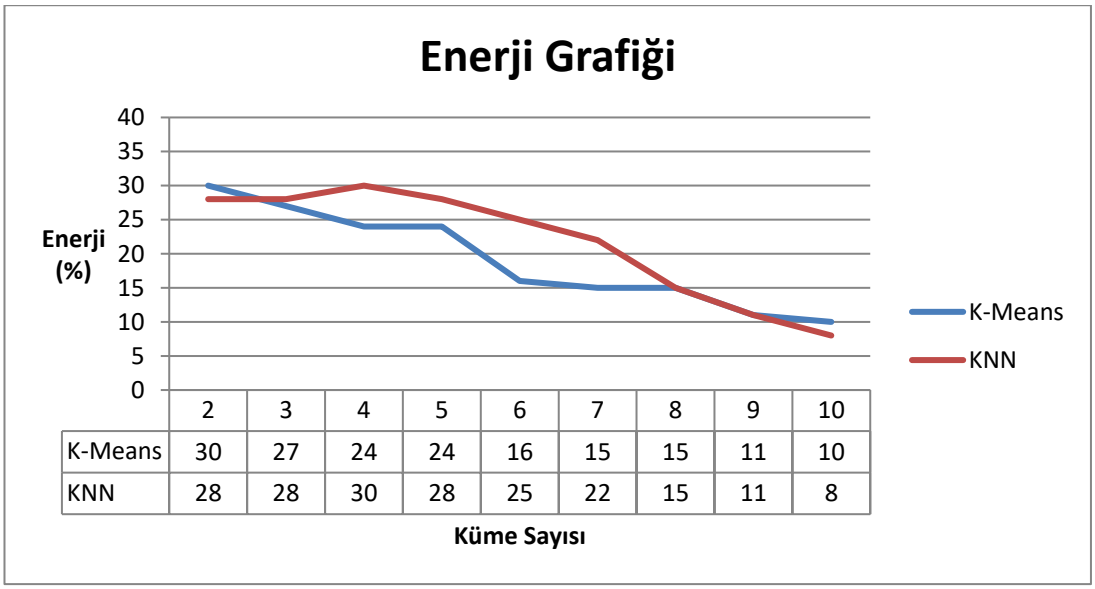
Piksel başına düşen renk derinliği ile ilgili olarak swift dilinde bir dezavantaj vardır. Bu durum şöyle açıklanabilir. Algoritma başladığında görüntü noktalarına ayrılıyor. Sıkıştırma bittikten sonra tekrar birleştirilmeye başlandığında swift diline özgü olan CGImageCreate komutunu kullanılır. CGImageCreate komutunun 11 inputundan bir tanesi bitsPerPixel inputudur. bitsPerPixel parametresi pixel başına düşen bit miktarını belirler. Algoritma ise sadece noktalarla çalıştığı için bitPerPixel değerinde bir değişme olmaz. Sürekli olarak 32Bit gerçek renk prensibiyle çalışır. Durum bu

şekilde olduğu için piksel başına düşen değer ile ilgili herhangi bir karşılaştırma yapılamamıştır.

Sıkıştırma oranı kriteri kullanılan resimlerin orijinal fiziksel boyutu ile resim gönderilmek için delta algoritması ile sıkıştırıldıktan sonra ortaya çıkan veri ile kıyaslanarak bulunmuştur. Yapay zekâ algoritmalarındaki resimlerde seçilecek rastgele merkez noktalarına ve kendi yapısı itibariyle farkların büyüklük küçüklüğüne göre aynı resimde dahi farklılık gösterebilir.

Lena resmi için yapılan incelemeler sonucunda alınan sonuçlar aşağıdaki grafiklerde görülmektedir.

Lena resmi için yapılan enerji testlerinde Şekil 5.2'de görüldüğü gibi küme sayısı arttıkça enerji ortalaması azalıyor. Küme sayısı ile enerji arasında ters bir orantı var. Buda demek oluyor ki gerek K-Means algoritması gerekse KNN algoritması bize bu işlem süresince enerji tasarrufu sağlamış oluyor.



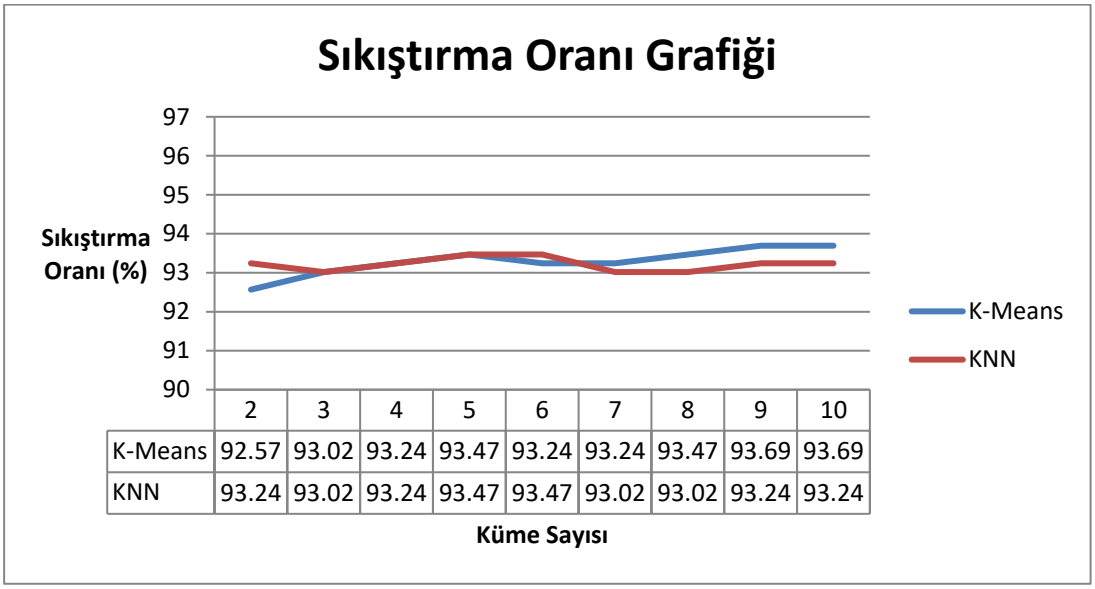
	2	3	4	5	6	7	8	9	10
K-Means	30	27	24	24	16	15	15	11	10
KNN	28	28	30	28	25	22	15	11	8

Şekil 5.2: Lena Resmi için Enerji Grafiği.

Lena resmi için K-Means algoritması ve KNN algoritması arasındaki karşılaştırmada, K-Means algoritması daha düşük küme miktarlarında enerji tasarrufu yapmaya başladığı ve küme sayısı arttıkça tasarruf etmeye devam ettiği sonuçları görülmüştür.

Lena resmi için Şekil 5.3'de bulunan sıkıştırma oranı grafiği göz önüne alındığında, sıkıştırma oran K-Means için ufak dalgalanmalarla da olsa bir artış gösteriyor. KNN

algoritması içinde durum çok değişik olmadığı görülüyor, fakat küme sayısı arttıkça KNN algoritması K-Means algoritmasının altında bir oranda seyrettiği görülmüştür.

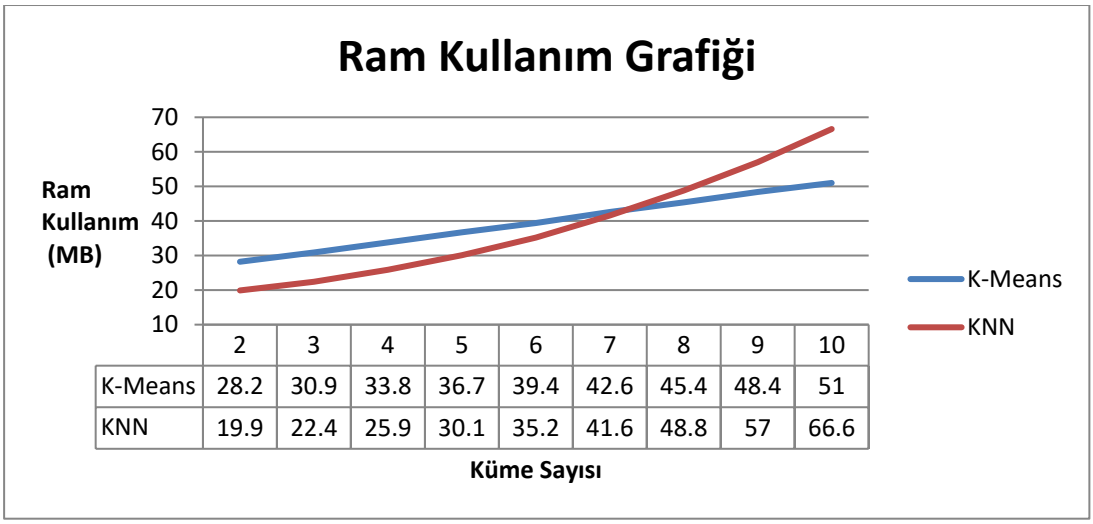


	2	3	4	5	6	7	8	9	10
K-Means	92.57	93.02	93.24	93.47	93.24	93.24	93.47	93.69	93.69
KNN	93.24	93.02	93.24	93.47	93.47	93.02	93.02	93.24	93.24

Şekil 5.3: Lena Resmi için Sıkıştırma Oranı Grafiği.

Küme sayıları dikkate alınmadığında, ortalama olarak bakıldığında bile %90'ları geçen sıkıştırma oranının yakalanması K-Means ve KNN algoritmalarını sıkıştırma işlemleri için kullanılabilineceği anlamına gelmiştir.

Ram kullanımı lena resmi için Şekil 5.4'te de görüldüğü gibi küme sayısı ile doğru orantılı bir davranış sergilemiştir. Küme sayısı arttıkça bellekte tutulan değişken ve döngüler arttığı için, sıkıştırma işlemi boyunca harcanan ram miktarı da artmıştır.

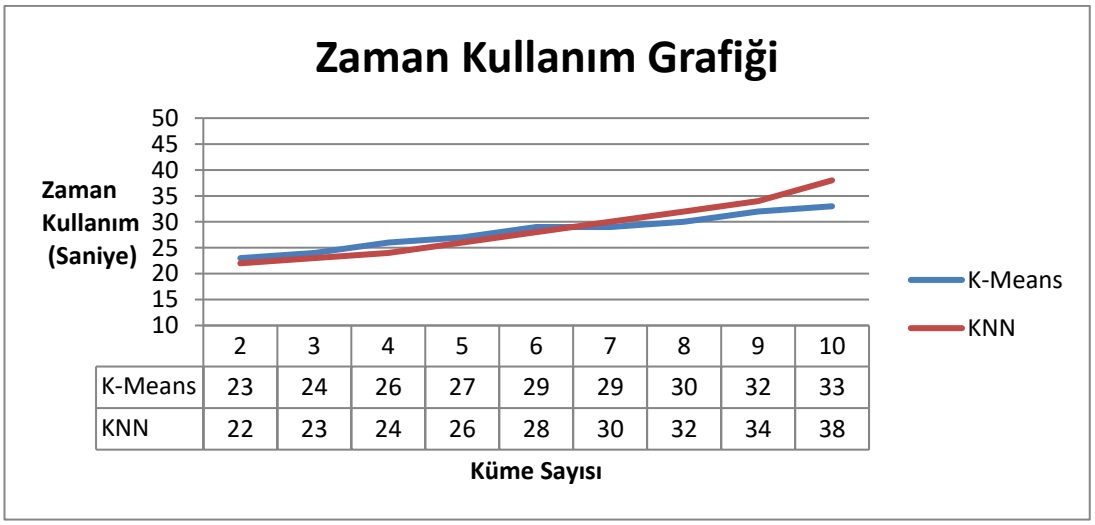


	2	3	4	5	6	7	8	9	10
K-Means	28.2	30.9	33.8	36.7	39.4	42.6	45.4	48.4	51
KNN	19.9	22.4	25.9	30.1	35.2	41.6	48.8	57	66.6

Şekil 5.4: Lena Resmi için Ram Kullanımı Grafiği.

K-Means algoritması için küme sayıları arasındaki fark aşağı yukarı sabit olarak görünse de bu durum KNN algoritması için aynı olmadığı saptanmıştır. KNN algoritmasında kümeler arası farklar açılarak ilerlediği için grafikte de belli bir küme sayısından sonra K-Means algoritmasından daha çok ram kullanmaya başlamıştır.

Zaman kullanımı lena resmi için Şekil 5.5'te de görüldüğü gibi küme sayısı ile doğru orantılı bir davranış sergilemiştir. Küme sayısı arttıkça bellekte tutulan değişken ve döngüler arttığı için, sıkıştırma işlemi boyunca harcanan ram miktarı ile doğru orantılı olarak artmıştır.



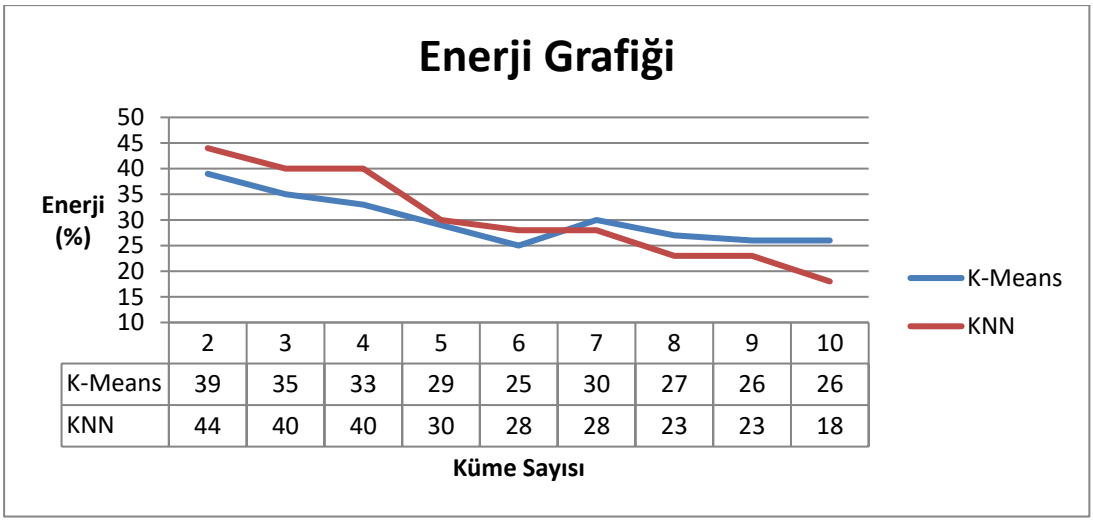
	2	3	4	5	6	7	8	9	10
K-Means	23	24	26	27	29	29	30	32	33
KNN	22	23	24	26	28	30	32	34	38

Şekil 5.5: Lena Resmi için Zaman Kullanımı Grafiği.

Pool resmi için yapılan incelemeler sonucunda alınan sonuçlar aşağıdaki grafiklerde görülmektedir.

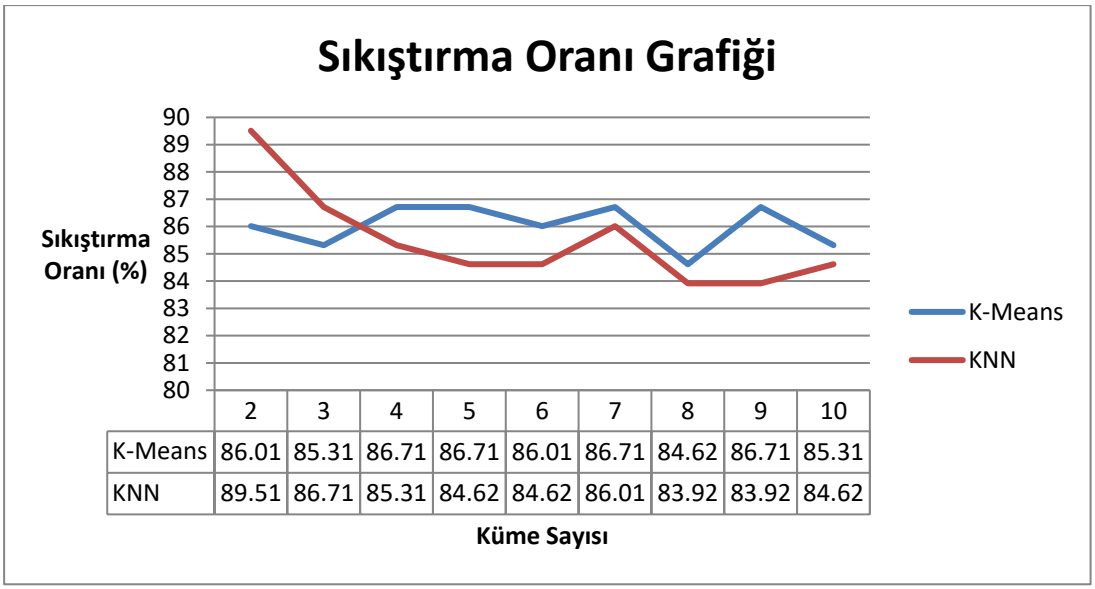
Pool resmi için Şekil 5.6'te bulunan Enerji grafiğine bakıldığında iki algoritma içinde dalgalı fakat azalan bir davranış göstermiştir.

İki algoritma arasında fark belli bir kümeden sonra K-Means algoritması enerji harcamasını biraz daha stabil hale getirmeye çalışmıştır. Bunun sebebi resimden alınan rastgele noktaların birbirlerine diğer resimlere oranla daha yakın olmasıdır.



Şekil 5.6: Pool Resmi için Enerji Grafiği.

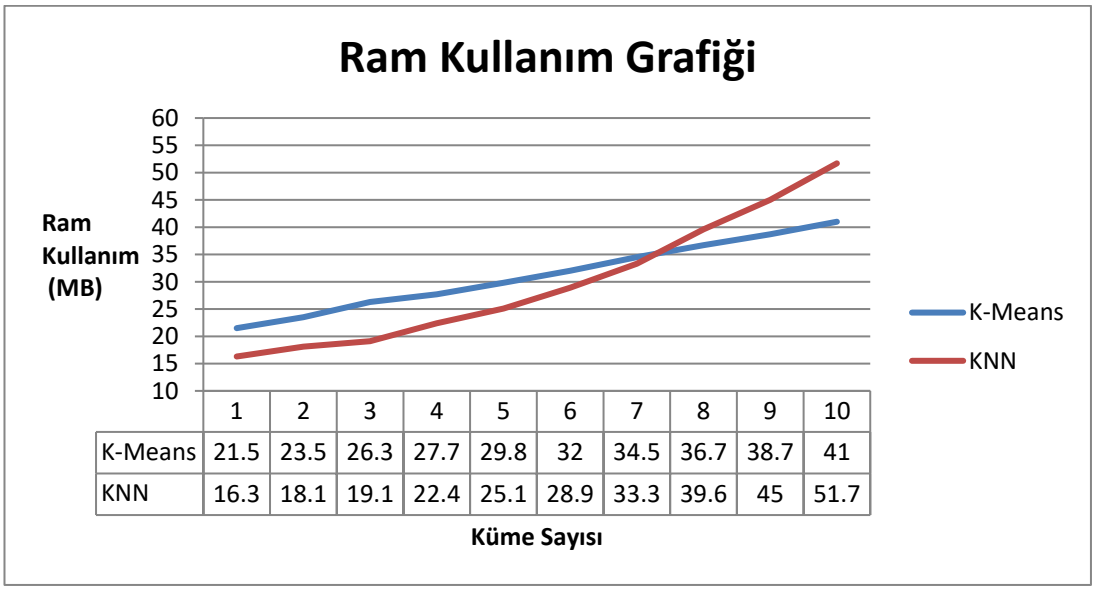
Pool resmi için sıkıştırma oranları Şekil 5.7'da bulunan grafik üzerinden incelendiğinde Lena'ya kıyasla %90'lardan %80'lere gerilemiştir. %80 sıkıştırma oranı da ciddi bir sıkıştırma oranıdır. Fakat Lena resmine göre küme sayısı arttıkça biraz daha dalgalı bir grafik oluşmuştur.



Şekil 5.7: Pool Resmi için Sıkıştırma Oranı Grafiği.

Pool resmi için ram kullanımı incelendiğinde Şekil 5.8'de görüldüğü üzere artan bir grafik mevcuttur. Küme sayısı arttıkça ram kullanımı da artar. Bu sonuç lena için yapılan testlerdeki sonuçlara çok benziyor. Bu sebeple bu durumdan ram kullanımı küme sayısıyla direkt alakalıdır fakat sıkıştırma ile dolaylı bir ilişkisi vardır.

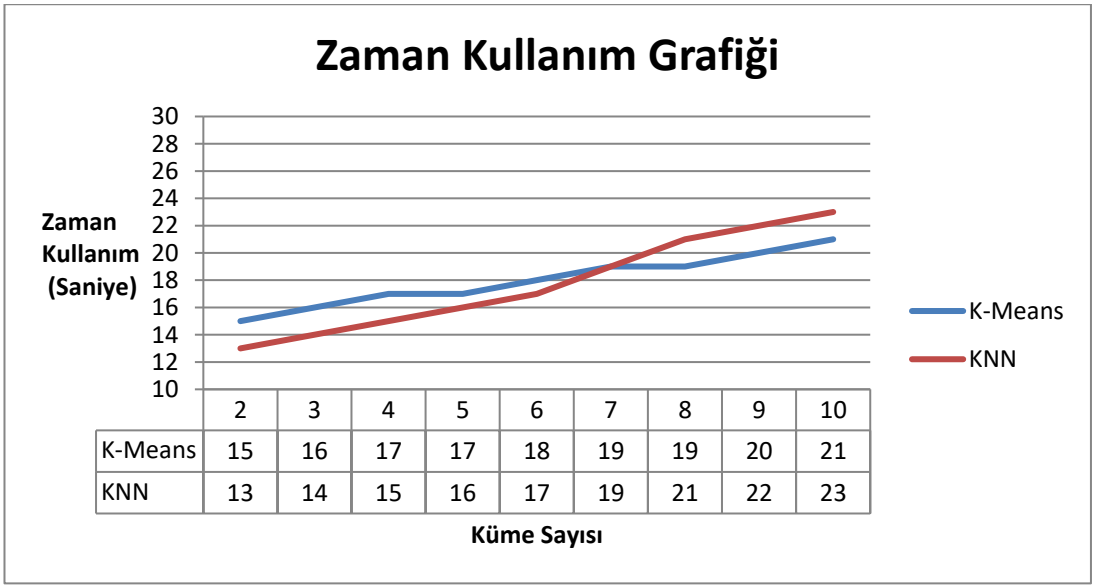
Algoritma bazında farklara bakıldığında K-Means algoritması yine sabit bir yol izlerken KNN algoritması belli bir küme sayısından sonra kümeler arası farkı açtığı için K-Means algoritmasından daha fazla ram kullanmaktadır.



	1	2	3	4	5	6	7	8	9	10
K-Means	21.5	23.5	26.3	27.7	29.8	32	34.5	36.7	38.7	41
KNN	16.3	18.1	19.1	22.4	25.1	28.9	33.3	39.6	45	51.7

Şekil 5.8: Pool Resmi için Ram Kullanım Grafiği.

Pool resmi için zaman kullanımı incelendiğinde Şekil 5.9'de görüldüğü üzere artan bir grafik mevcuttur. Küme sayısı arttıkça zaman kullanımı da artar. Ram kullanımı ile doğru orantılıdır. Bu sonuç lena için yapılan testlerdeki sonuçlara çok benziyor.

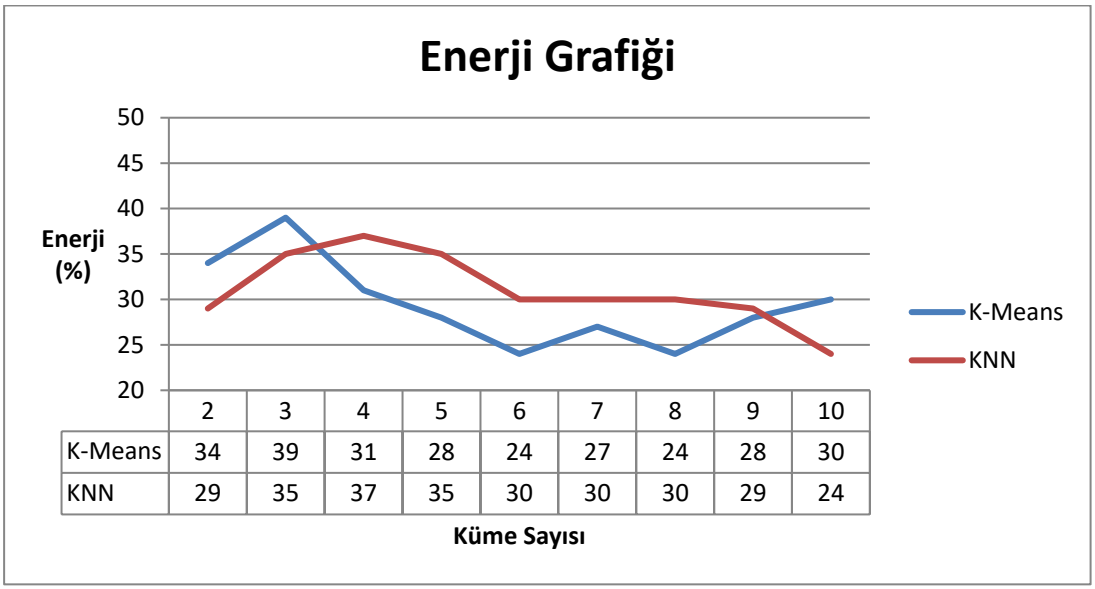


	2	3	4	5	6	7	8	9	10
K-Means	15	16	17	17	18	19	19	20	21
KNN	13	14	15	16	17	19	21	22	23

Şekil 5.9: Pool Resmi için Zaman Kullanım Grafiği.

Cat resmi için yapılan incelemeler sonucunda alınan sonuçlar aşağıdaki grafiklerde görülmektedir.

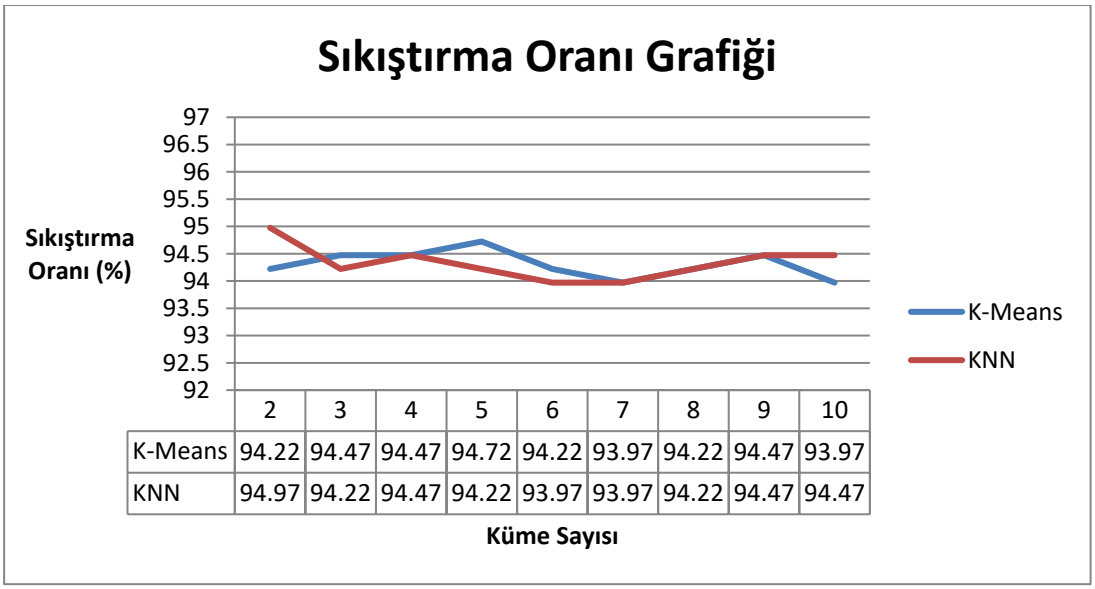
Şekil 5.10'daki Car resmi için Enerji grafiği incelendiğinde, yapılan testlerde K-Means algoritması belli bir kümenin üstünde KNN algoritmasının üstüne çıksa da ortalamaya bakıldığında KNN algoritmasından daha fazla enerji tasarrufu gerçekleştirmektedir.



	2	3	4	5	6	7	8	9	10
K-Means	34	39	31	28	24	27	24	28	30
KNN	29	35	37	35	30	30	30	29	24

Şekil 5.10: Cat Resmi için Enerji Grafiği.

Cat resmi için sıkıştırma oranına bakıldığında Şekil 5.11'de görüldüğü gibi tekrar %90'larda bir sıkıştırma oranı görülüyor.

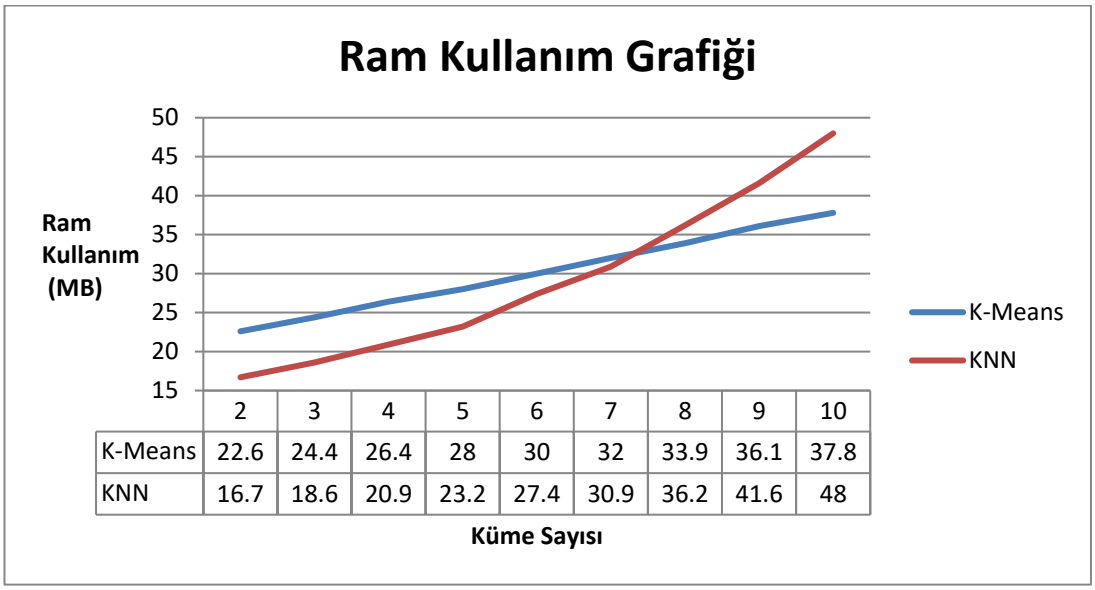


	2	3	4	5	6	7	8	9	10
K-Means	94.22	94.47	94.47	94.72	94.22	93.97	94.22	94.47	93.97
KNN	94.97	94.22	94.47	94.22	93.97	93.97	94.22	94.47	94.47

Şekil 5.11: Cat Resmi için Sıkıştırma Oranı Grafiği.

Lena resmi ile çok benzer bir grafik ortaya çıkmış oluyor. İki algoritmayı karşılaştıracak olursan bu resim için küme değerlerindeki farkları çok küçük olsa da ortalama seyirler aynı düzeyde ilerliyor.

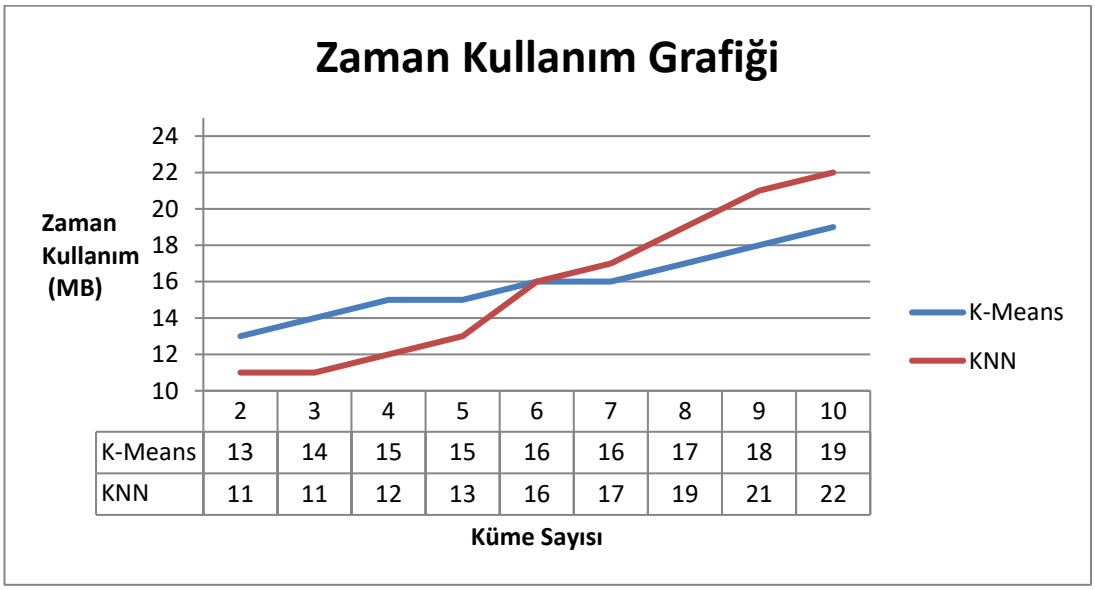
Cat resmi için ram kullanımı incelendiğinde diğer resimlerle çok benze bir sonuç alındığı Şekil 5.12'deki grafikte görünüyor. Bununla beraber ram kullanımının durumu sabitlenmiş oluyor.



	2	3	4	5	6	7	8	9	10
K-Means	22.6	24.4	26.4	28	30	32	33.9	36.1	37.8
KNN	16.7	18.6	20.9	23.2	27.4	30.9	36.2	41.6	48

Şekil 5.12: Cat Resmi için Ram Kullanımı Grafiği.

Cat resmi için zaman kriteri incelendiğinde diğer resimlerle benzer bir sonuç alındığı Şekil 5.13'de görünüyor. Bununla beraber zaman kriteri durumu sabitlenmiş oluyor.

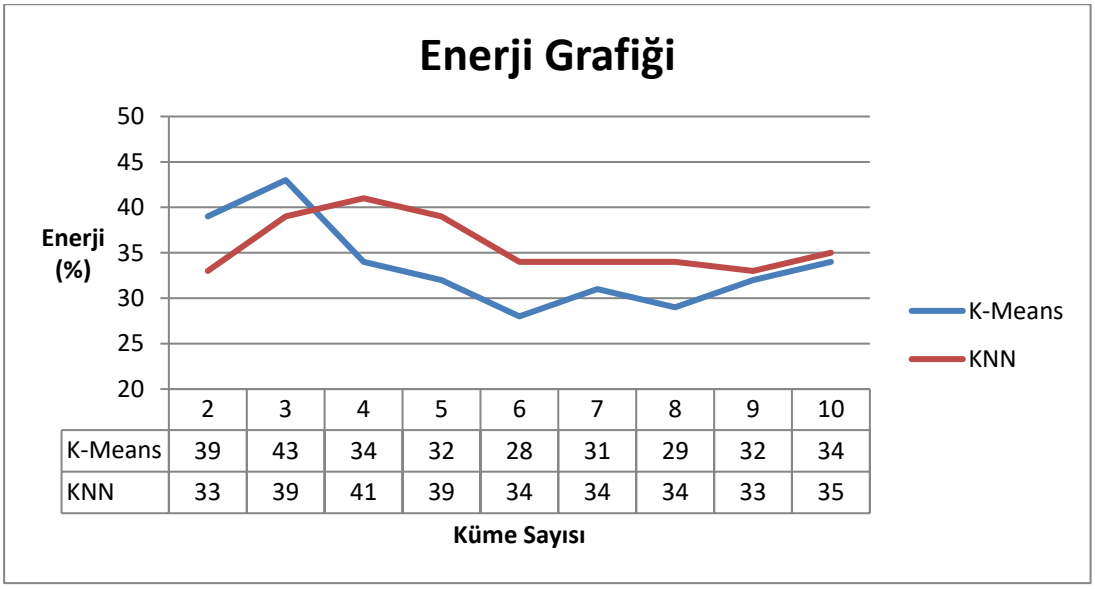


	2	3	4	5	6	7	8	9	10
K-Means	13	14	15	15	16	16	17	18	19
KNN	11	11	12	13	16	17	19	21	22

Şekil 5.13: Cat Resmi için Zaman Kullanımı Grafiği.

Barbara resmi için yapılan incelemeler sonucunda alınan sonuçlar aşağıdaki grafiklerde görülmektedir.

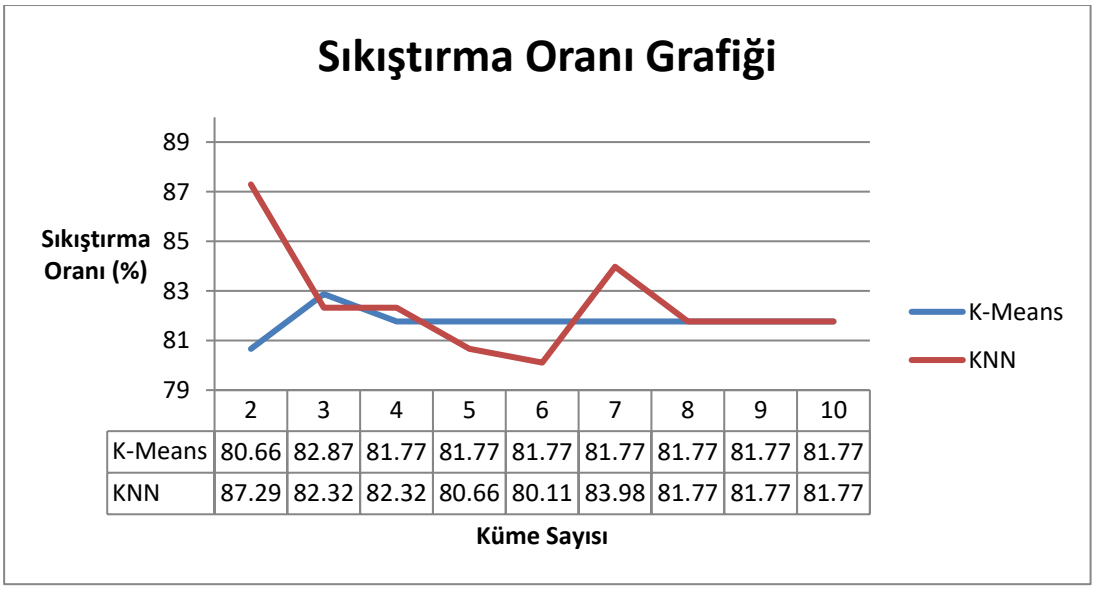
Şekil 5.14'deki Barbara resmi için Enerji grafiği incelendiğinde, yapılan testlerde K-Means algoritması belli bir kümede KNN algoritmasının üstüne çıksa da ortalamaya bakıldığında KNN algoritmasından daha fazla enerji tasarrufu gerçekleştirmektedir.



	2	3	4	5	6	7	8	9	10
K-Means	39	43	34	32	28	31	29	32	34
KNN	33	39	41	39	34	34	34	33	35

Şekil 5.14: Barbara Resmi için Enerji Grafiği.

Barbara resmi için sıkıştırma oranına bakıldığında Şekil 5.15'de görüldüğü gibi %80'lerde bir sıkıştırma oranı görülüyor. Renkler birbirine zıt olduğu sıkıştırma oranı diğer resimlere göre düşüş göstermiştir.



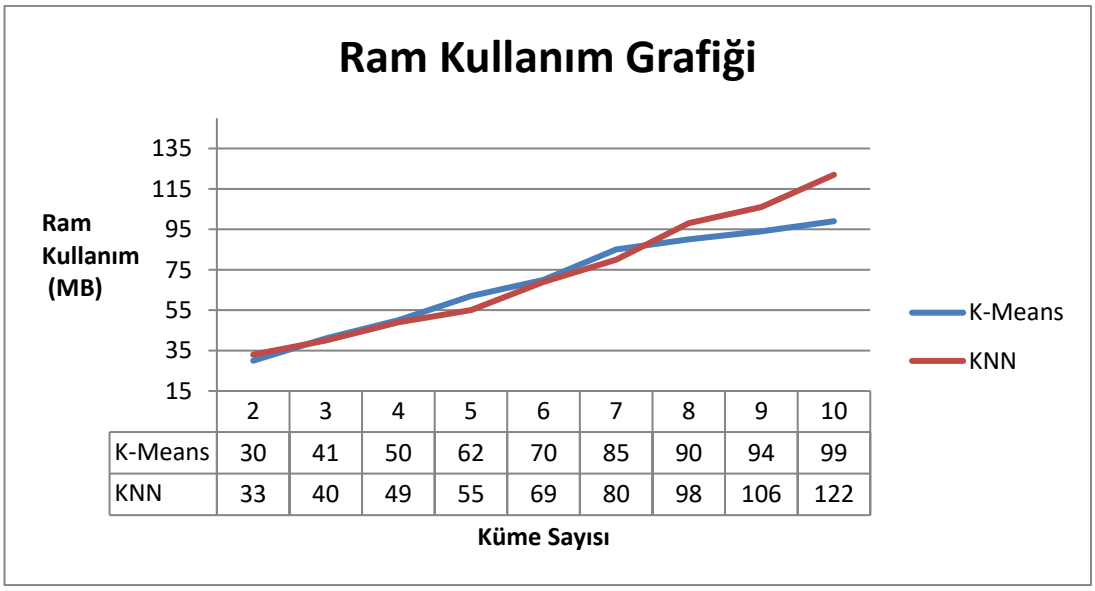
	2	3	4	5	6	7	8	9	10
K-Means	80.66	82.87	81.77	81.77	81.77	81.77	81.77	81.77	81.77
KNN	87.29	82.32	82.32	80.66	80.11	83.98	81.77	81.77	81.77

Şekil 5.15: Barbara Resmi için Sıkıştırma Oranı Grafiği.

İki algoritmayı karşılaştıracak olursak bu resim için küme değerlerindeki farkları çok küçük olsa da ortalama seyirler aynı düzeyde ilerliyor.

Barbara resmi için ram kullanımı incelendiğinde diğer resimlerle çok benzer bir sonuç alındığı Şekil 5.16'deki grafikte görünüyor. Küme sayısı arttıkça KNN Algoritması daha fazla ram kullanmaya başlıyor.

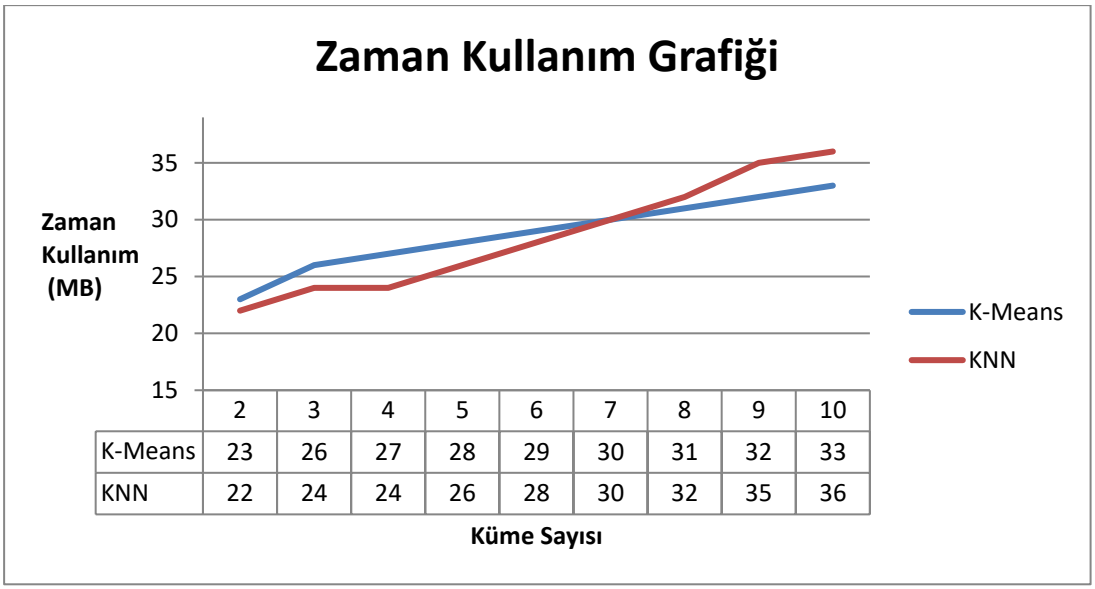
Resmin siyah beyaz yani gri tonlamalı olması ram kullanımına bir farklı bir etken olarak görünmüyor. Renkli resimler gibi ram kullanımı tamamen yapılan küme sayısıyla alakalı bir kriter.



	2	3	4	5	6	7	8	9	10
K-Means	30	41	50	62	70	85	90	94	99
KNN	33	40	49	55	69	80	98	106	122

Şekil 5.16: Barbara Resmi için Ram Kullanımı Grafiği.

Barbara resmi için ram kullanımı incelendiğinde diğer resimlerle çok benzer bir sonuç alındığı Şekil 5.17'deki grafikte görünüyor. Bununla beraber zaman kullanımının durumu sabitlenmiş oluyor. Gri tonlamalı bir resmin kullanılması zaman kullanımını arttırmıştır. Bu durum renkler arasındaki farklarla alakalıdır.

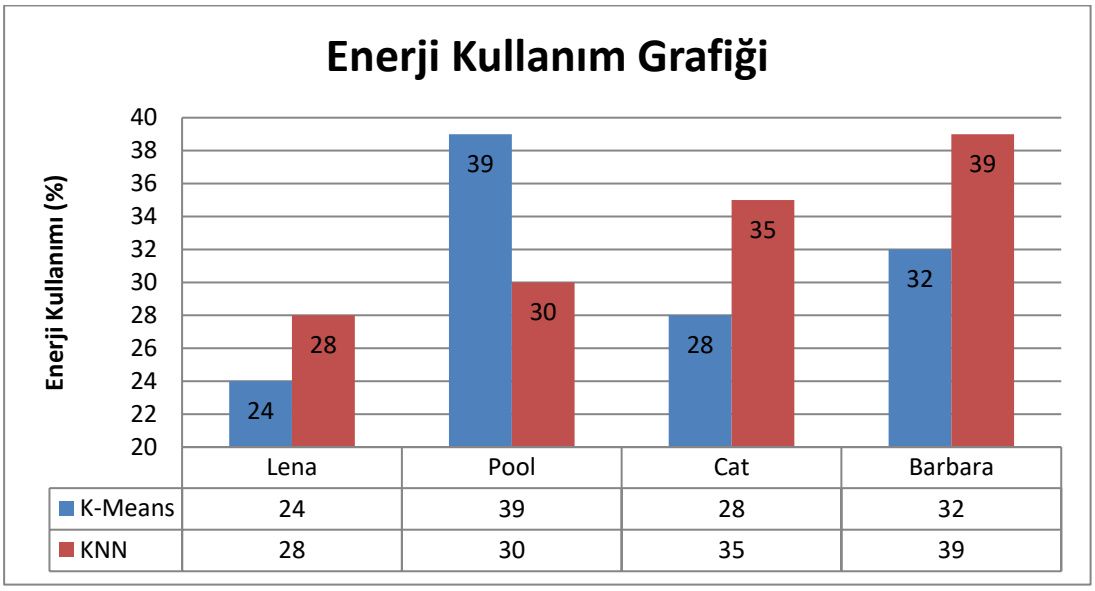


	2	3	4	5	6	7	8	9	10
K-Means	23	26	27	28	29	30	31	32	33
KNN	22	24	24	26	28	30	32	35	36

Şekil 5.17: Barbara Resmi için Zaman Kullanımı Grafiği.

Kullanılan 4 resmin birbirleriyle karşılaştırılması için küme sayısı 5 kabul edilmiştir ve bu değere göre enerji kullanım, sıkıştırma oranı, ram kullanım ve zaman kullanım grafikleri oluşturulmuştur.

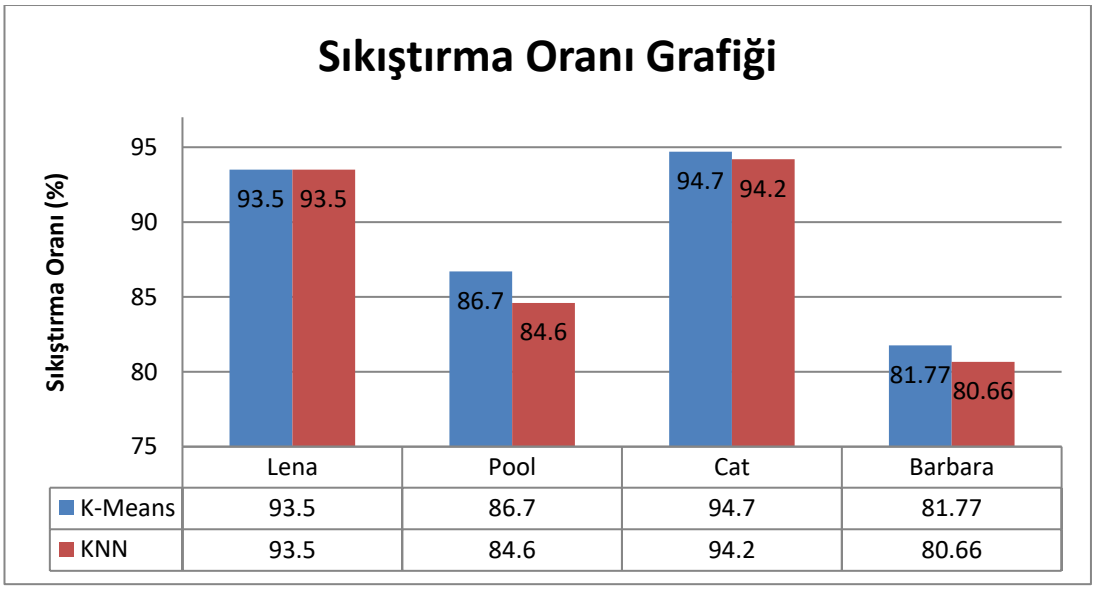
Şekil 5.18'e bakıldığında algoritmalar arası resimler bazında ne kadar enerji harcandığı gösterilmiştir. Bu grafiğe göre KNN algoritmasının daha fazla enerji harcadığını fakat Pool resminde istisnai bir biçimde resme bağlı olarak K-Means algoritmasından daha az enerji harcadığı görülmüştür.



	Lena	Pool	Cat	Barbara
K-Means	24	39	28	32
KNN	28	30	35	39

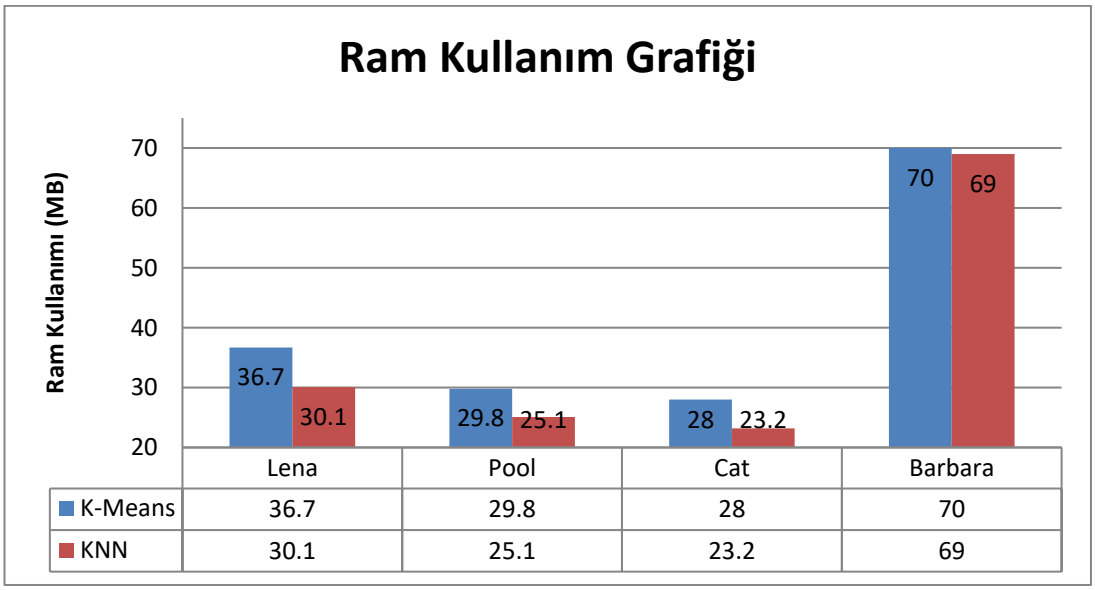
Şekil 5.18: Test resimlerinin enerji kullanımı bazında karşılaştırılması.

Şekil 5.19'deki resimler ve algoritmalar arası sıkıştırma oranları görüldüğünde resim bazında fiziksel olarak algoritmalar arası farkın çok az olduğu gözlemlenmiştir. İhtiyacı saklama alanı olarak belirlenen bir çalışmada K-Means ve KNN algoritması arasında seçil yapılması mantıklı olmayacaktır.



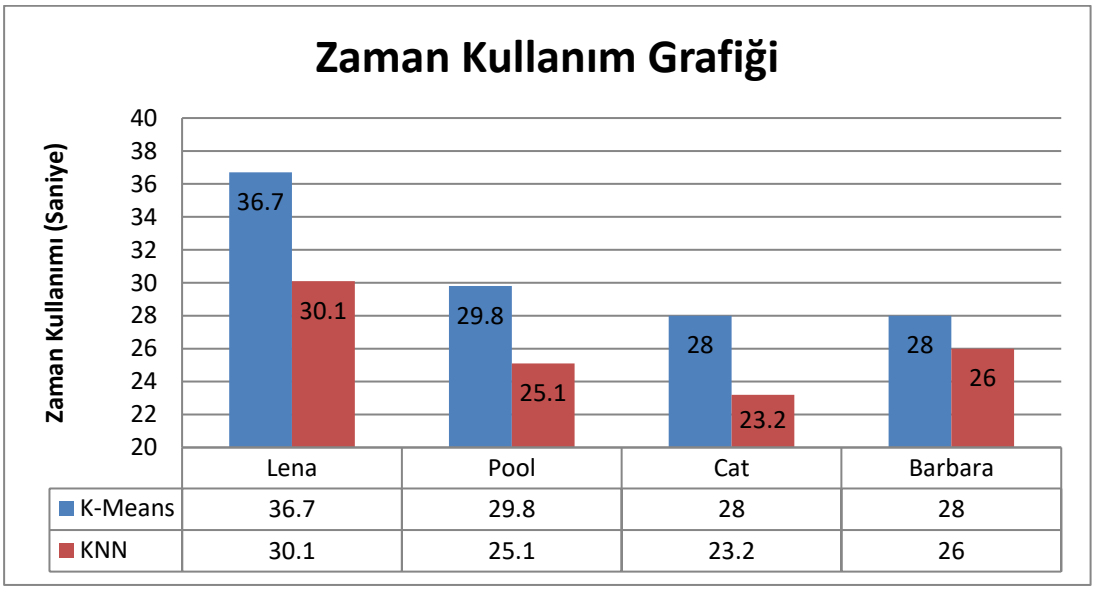
Şekil 5.19: Test resimlerinin sıkıştırma oranı bazında karşılaştırılması.

Şekil 5.20'de Küme sayısı 5 olan test sonuçları için K-Means algoritmasının tüm resimlerde daha fazla ram kullanıldığı sonucuna varılmıştır. Yapılacak çalışmalar için önemli olan ram performansı ise KNN kullanılması daha mantıklı olacaktır.



Şekil 5.20: Test resimlerinin ram kullanım bazında karşılaştırılması.

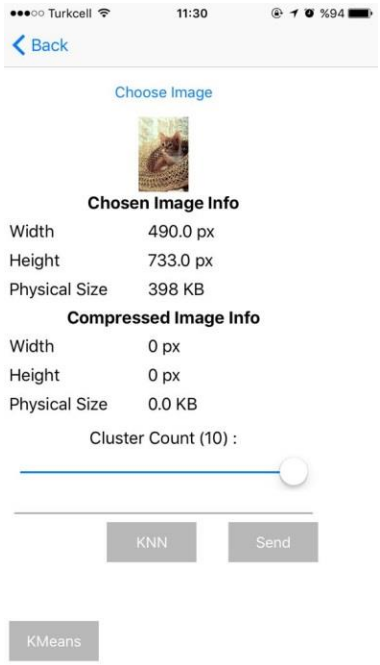
Şekil 5.21'de Küme sayısı 5 olan test sonuçları için K-Means algoritmasının tüm resimlerde daha fazla zaman kullanıldığı sonucuna varılmıştır. Yapılacak çalışmalar için önemli olan zaman performansı ise KNN kullanılması daha mantıklı olacaktır.



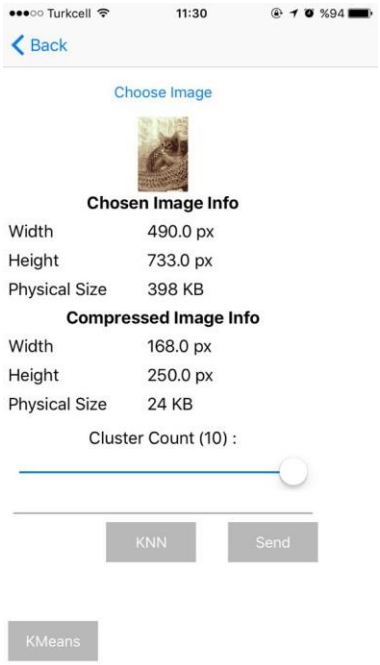
Şekil 5.21: Test resimlerinin zaman kullanım bazında karşılaştırılması.

Test resimlerinin sıkıştırılmadan önceki ve sonraki durumlarını gösteren örnekler verilmiştir. Resim bazında görüntünün bozulup bozulmadığına subjektif olarak yardımcı olabilir.

Şekil 5.22 ve 5.23'te Cat resminin sıkıştırılmadan önceki ve sonraki resimleri görülmektedir. Resimlerde de görüldüğü üzere resmin fiziksel boyutlarında bir değişim olmasına rağmen subjektif olarak görselde bir bozulma söz konusu değildir.

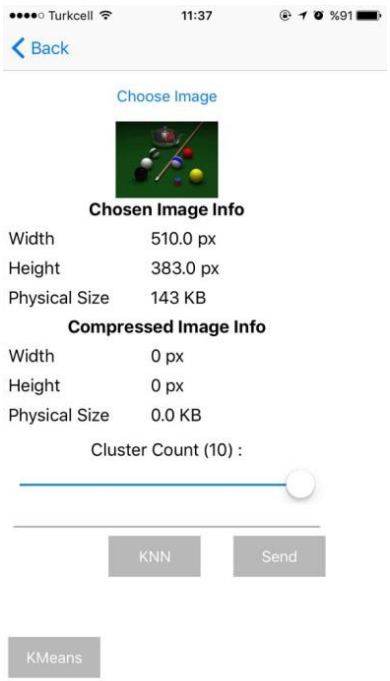


Şekil 5.22: Cat resminin sıkıştırılmadan önceki durumu.

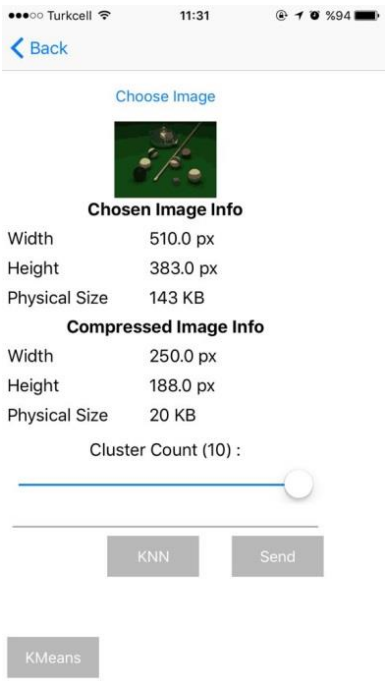


Şekil 5.23: Cat resminin sıkıştırıldıktan sonraki durumu.

Şekil 5.24 ve 5.25'te Pool resminin sıkıştırılmadan önceki ve sonraki resimleri görülmektedir. Pool resminde subjektif olarak bozulma mevcuttur. Fakat diğer kriterlerde ki değişim hala olumlu olarak devam etmektedir.

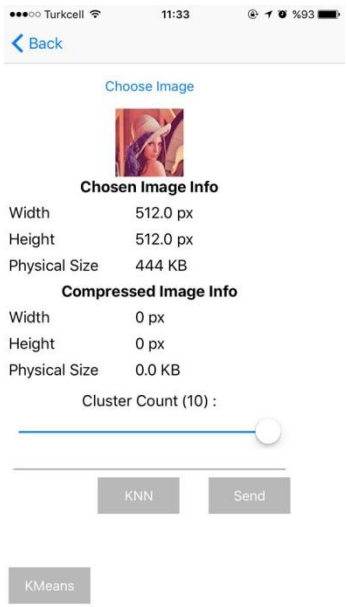


Şekil 5.24: Pool resminin sıkıştırılmadan önceki durumu.

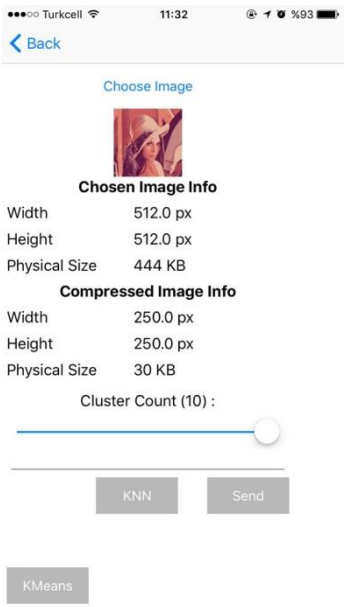


Şekil 5.25: Pool resminin sıkıştırıldıktan sonraki durumu.

Şekil 5.26 ve 5.27'de Lena resminin sıkıştırılmadan önceki ve sonraki durumu görülmektedir. Lena'da da Cat resmindeki gibi görsel bozukluk mevcut değildir.

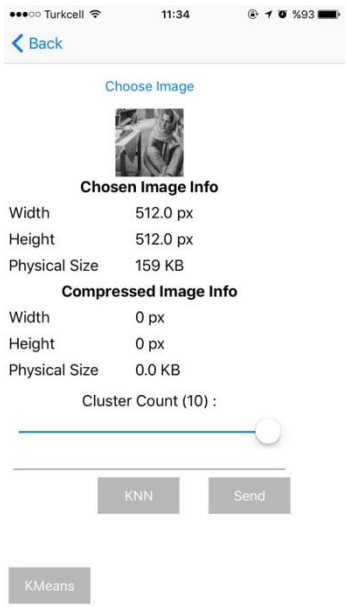


Şekil 5.26: Lena resminin sıkıştırılmadan önceki durumu.

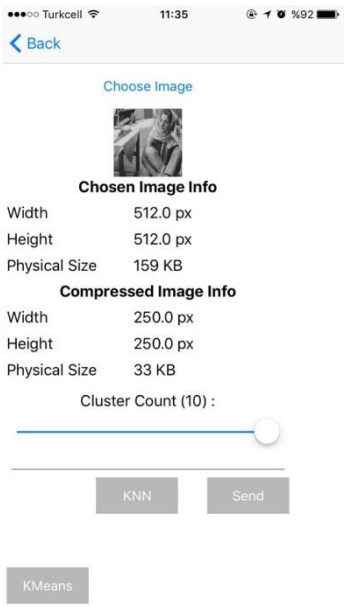


Şekil 5.27: Lena resminin sıkıştırıldıktan sonraki durumu.

Şekil 5.28 ve 5.29'da Barbara resminin sıkıştırılmadan önceki ve sonraki resimleri görülmektedir. Barbara resminde diğer resimlerden farklı olarak siyah beyaz resim olmasına bağlı görsel bir bozulma mevcut değildir.



Şekil 5.28: Barbara resminin sıkıştırılmadan önceki durumu.



Şekil 5.29: Barbara resminin sıkıştırıldıktan sonraki durumu.

6. SONUÇ

Yapılan araştırma sonucunda elde ettiğimiz verilere göre sıkıştırma sonuçlarına etki eden kriter sayısı çok fazladır. Fakat buna rağmen içlerinde bazı kriterler incelendiğinde istenilen sonuçlara ulaşabiliriz.

Enerji konusu ele alındığında, tek kümeli resimler haricinde, küme sayısı arttıkça resim ayırmaksızın ram kullanımı ve buna bağlı olarak toplam enerji kullanımı artmıştır. Fakat aynı şartlarda ortalama enerji kullanımı düşmeye başlamıştır. Bu da küme sayısı arttıkça, başlangıçta yapılan sabit maliyetin kümelere dağıtıldığını gösterir.

Ram kullanımı incelendiğinde, küme ayırmaksızın artışla doğru orantılı olarak artan grafikler görülmüştür. Küme sayısı arttıkça yapılan işlemlerin artması, daha çok değişken ihtiyacı ve döngü doğal olarak ram kullanımını arttırmıştır.

Orijinal resme kıyasla, algoritmaların ortaya çıkardığı sıkıştırılmış resim %80 ile %96 arasında daha küçük fiziksel boyutlara ulaşıyor. Bu değerler yine renk kümelerinin artmasıyla beraber düşüş göstermiştir.

Sonuç olarak sıkıştırma işleminde en önemli etken küme sayısı olarak görülmüştür. Yüksek sıkıştırma oranları her ne kadar olumluymuş gibi görünse de bu oranların bu kadar yüksek olması resmin çok küçük olmamasına bağlıdır. Küçük resimlerde sıkıştırma oranı düşebilir.

Özetle eğer sıkıştırma işlemi yaparken kullanılan resmin boyutları büyükse her iki algoritma da sıkıştırma işlemi için uygundur.

Bunun yanında resmin bozulmaması öncelikli ise siyah beyaz resimlerde daha az bozulma olduğu simülasyon sonuçlarında görülmüştür.

Sıkıştırma işlemi yaparken cihazın kaynak kullanımı öncelikli ise KNN algoritmasının daha az kaynak yani ram kullandığı simülasyon sonuçlarında görülmüştür.

Sıkıştırma işlemi yaparken önemli olan zaman kriteriyse yine kaynak kullanımında olduğu gibi KNN algoritması seçilmesi daha mantıklı olacaktır.

IOS cihazlar için yapay zeka ve delta algoritması ile görüntü sıkıştırıp sohbet ortamında paylaştıkları bu uygulamanın, yapılan performans ve verimlilik

analizlerine dayanarak, ileriye dönük bu araştırmayı geliştirmek amacıyla, yapılabilecekler arasında şu öneriler yer alabilir:

- Bu araştırmada kullanılan K-Means ve KNN algoritmaları yerine bunların dışında belirlenen bir algoritma kullanılabilir.
- Socket.IO geliştirilerek görüntü string olarak değilde upload yoluyla kullanıcılara direk resim olarak gönderilebilir. Bu şekilde uygulamayı kullanan kullanıcı sayısı arttığında ve buna bağlı olarak gönderilen görüntü trafiği arttığında, delta decompress işleminden kaynaklı oluşması muhtemel yavaşlıklar önlenmiş olur.
- Uygulamayı kullanacak kullanıcıların sayısını kısıtlamamak için IOS cihazlara ek olarak Android ve Windows Uygulamaları yazılabilir. Böylece platform kısıtı olmadan neredeyse tüm cihazlarda, ortaya çıkan ürün kullanılabilecektir.

KAYNAKLAR

[1] İMAMOĞLU Kadir, **Görüntü sıkıştırma ve internet üzerinden güvenli aktarım sistemi**, Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, Kocaeli

[2] AYBAR Müjdat, **Yapay sinir ağlarının dijital görüntü sıkıştırılmasında kullanımı**, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, İzmir

[3] ÜNAL Öner, **Yapay sinir ağları ile görüntü sıkıştırma ve görüntü kütük biçimi**, Hacettepe Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, İstanbul

[4] KERPİÇ Uğur, **Yapay sinir ağları kullanılarak görüntü sıkıştırma**, Yeditepe Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2005

[5] BAYDOĞAN Emre, **Ağlarda adaptif veri sıkıştırma: Öğrenebilen durum makinesi yaklaşımı**, Marmara Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2001

[6] MOTLAGHReza H.,**Tekrarlamalı fonksiyon sistemlerinin fraktal teorisi üzerinde kurulu görüntü kompresyonu**, Orta Doğu Teknik Üniversitesi Fen Bilimleri Üniversitesi, Yüksek Lisans, 1995

[7] BAĞLAN Sezer, **JPEG XR başarımı, İstanbul Teknik Üniversitesi Fen Bilimleri Üniversitesi**, Yüksek Lisans, 2014

[8] KILIÇ İlker, **Dijital görüntü sıkıştırma teknikleri**, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1997

[9] KATKAR Ali, **Biyomedikal görüntülerin dalgacık dönüşümü ile sıkıştırılması**, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2002

[10] BİLGİN Gökhan, **Dalgacık dönüşümleri ve hiyerarşik ağaçlarda küme bölümleme yöntemi ile görüntü sıkıştırma**, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2003

[11] TÜRK Akın, **Uzaysal yöntem kullanarak görüntü verisi sıkıştırması**, Orta Doğu Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1993

[12] DİNÇ Murat, **Fractal Image Compression**, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1996

[13] ERSOY İlker, **Parmakizi görüntülerinin model tabanlı yaklaşımla sıkıştırılması**, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1996

[14] TOSUN Mustafa, **Geriyansımalı yapay sinir ağının görüntü sıkıştırmada kullanılması**, Dumlupınar Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1997

[15] ÖZDEMİR Osman, **Radyal taban fonksiyonlu yapay sinir ağları ile fraktal görüntü sıkıştırma**, İstanbul Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1998

[16] ÖNGEN Berna, **Veri sıkıştırma yöntemleri**, Dokuz Eylül Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1998

[17] KASAPOĞLU Yusuf, **8 bit gri tonlu görüntü sıkıştırma uygulamaları**, Gebze İleri teknoloji Enstitüsü, Mühendislik ve Fen Bilimleri Enstitüsü, Yüksek Lisans, 1998

[18] TÜRK Mustafa, **Görüntü sıkıştırma algoritmaları**, Fırat Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 1998

[19] SARAL Tunç, **Wavelet transformasyonları ile görüntü sıkıştırma**, Yıldız Teknik Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2000

[20] BAHADIR Ali, **Çok tabakalı (MLP) yapay sinir ağı ile görüntü işleme ve görüntü sıkıştırma**, Niğde Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2001

[21] KURNAZ Çetin, **Görüntü işlemede kullanılan DCT-DWT yöntemlerinin karşılaştırılması ve bu yöntemlerin incelenmesi için test görüntülerinin oluşturulması**, Ondokuz Mayıs Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2002

[22] YAVUZ Hasan Serhan, **Bulanık görüntü bölütleme yöntemiyle ikinci nesil görüntü sıkıştırma**, Eskişehir Osmangazi Üniversitesi Fen Bilimleri Enstitüsü, Yüksek Lisans, 2002

[23] ALAYBEYOĞLU Ayşegül, **A DISTRIBUTED IMAGE COMPRESSION ALGORITHM FOR ENVIRONMENTAL MONITORING APPLICATIONS**, Aralık2013, Electronics World; Dec2013, Vol. 119 Issue 1932, p26

[24] ALAYBEYOĞLU Ayşegül, **A Distributed Image Compression Algorithm Based on a Genetic Algorithm Based Clustering Infrastructure for Mobile Wireless Multimedia Sensor Networks,** Ekim 2013, Life Sci J 2013;10(4):2908-2914], (ISSN:1097-8135)

[25] Artificial Intelligence, Britannica, global.britannica.com/technology/artificial-intelligence, (SGT: 12.01.2017)

[26] ČAPEK Karel, **R.U.R**, Mart 1970, Pocket Books ISBN 0-671-46605-4

[27] Rosenfeld Azriel, **Picture Processing by Computer**, Temmuz 1973, DOI: 10.1126/science.169.3941.166

[28] MESUT Altan, **Veri Sıkıştırmada Yeni Yöntemler,** Trakya Üniversitesi Fen Bilimleri Enstitüsü, Doktora,2006

[29] MacQueen, J.,**Some methods for classification and analysis of multi-variate observations**. In: Proc. oftheFifth Berkeley Symp. on Math.,Statistics and Probability, LeCam, L.M., and Neyman, J., (eds.), Berkeley: U. California Press, 281. 1967

[30] Tom M. Mitchell, **Machine Learning**, Mart 1997, ISBN: 0.070.428.077

[31] Nikolai, **Picture Processing by Computer**, Ekim 2012, DOI: 10.1126/science.169.3941.166

[32] iOS, Wikipedia, https://tr.wikipedia.org/wiki/%C4%B0OS, (SGT: 11.01.2017).

[33] Node.JS, Wikipedia, https://tr.wikipedia.org/wiki/Node.js, (SGT: 11.01.2017).

[34] Economic & Social Commission for Western Asia, **Review of Science and Technology in ESCWA Member Countries,** September 2001, ISBN: 92-1-128234-

[35] DUYAR Utku, "Dijital Görüntü Teknolojileri", Elektrik Mühendisliği Dergisi, Kasım 2010, s.17-18.

[36] Bilgin, T. T.: "Çok Boyutlu Uzayda Görsel Veri Madenciliği İçin Üç Yeni Çatı Tasarımı Ve Uygulamaları", Doktora Tezi, Marmara Üniversitesi Fen

Bilimleri Enstitüsü, İstanbul, (2007).

[37] SARAÇ Ömer Faruk, "Yapay Sinir Ağları Ve K-Means Kullanarak Sınır Değerlerine Göre Yazılım Efor Tahmini", Yüksek Lisans Tezi, Kocaeli Üniversitesi Fen Bilimleri Enstitüsü, 2014

[38] Socket.io, http://socket.io/, (SGT: 11.01.2017).

[39] Delta encoding, https://en.wikipedia.org/wiki/Delta_encoding, (SGT: 11.01.2017).

EKLER

EK A: Bu teze konu olan IOS Tabanlı Mobil Cihazlar İçin Yapay Zeka Yöntemi İle Görüntü Sıkıştırma Ve İletim Uygulamasının, ana fikrini oluşturan bazı metotların kod çıktıları aşağıda verilmiştir. IOS programlamaya örnek olması açısından incelenmesinde fayda vardır.

askForNickname metodu uygulamanın girişindeki kullanıcı adı isteme metodu.

```
funcaskForNickname() {
letalertController = UIAlertController(title: "GO Chat", message: "Pleaseenter a
nickname:", preferredStyle: UIAlertControllerStyle.Alert)

alertController.addTextFieldWithConfigurationHandler(nil)

letOKAction = UIAlertAction(title: "OK", style: UIAlertActionStyle.Default) {
(action) ->Void in
lettextfield = alertController.textFields![0]
if textfield.text?.characters.count == 0 {
self.askForNickname()
}
else {
self.nickname = textfield.text

SocketIOManager.sharedInstance.connectToServerWithNickname(self.nickname
, completionHandler: { (userList) ->Void in
dispatch_async(dispatch_get_main_queue(), { () ->Void in
ifuserList != nil{
self.users = userList
self.tblUserList.reloadData()
self.tblUserList.hidden = false
}
                })
            })
}
}

alertController.addAction(OKAction)
presentViewController(alertController, animated: true, completion: nil)
}
```

handleUserTypingNotification metodu yazmaya başlanıldığında yazı yazıldığını sunucuya bildiren aynı zamanda diğer kullanıcılara da bildirimi gösteren metot.

```
funchandleUserTypingNotification(notification: NSNotification) {
iflettypingUsersDictionary = notification.object as? [String: AnyObject] {
varnames = ""
vartotalTypingUsers = 0
for (typingUser, _) in typingUsersDictionary{
iftypingUser != nickname{
names = (names == "") ? typingUser : "\(names), \(typingUser)"
totalTypingUsers += 1
}
}

iftotalTypingUsers> 0 {
letverb = (totalTypingUsers == 1) ? "is" : "are"

lblOtherUserActivityStatus.text = "\(names) \(verb) nowtyping a message..."
lblOtherUserActivityStatus.hidden = false
}
else {
lblOtherUserActivityStatus.hidden = true
}
}
}
```

handleKeyboardDidShowNotification metodu yazma alanına tıklandığında klavyeyi gösteren metot.

```
funchandleKeyboardDidShowNotification(notification: NSNotification) {
ifletuserInfo = notification.userInfo {
ifletkeyboardFrame = (userInfo[UIKeyboardFrameBeginUserInfoKey] as?
NSValue)?.CGRectValue() {
conBottomEditor.constant = keyboardFrame.size.height
view.layoutIfNeeded()
}
}
}
```

tableView, metin veya resim mesajlarının listelenmesini sağlayan metodudur.

```
functableView(tableView: UITableView, cellForRowAtIndexPathindexPath:
NSIndexPath) ->UITableViewCell{
letcell = tableView.dequeueReusableCellWithIdentifier("idCellChat",
forIndexPath: indexPath) as! ChatCell
letcurrentChatMessage = chatMessages[indexPath.row]
letsenderNickname = currentChatMessage["nickname"] as! String
letmessage = currentChatMessage["message"] as! String
letmessageDate = currentChatMessage["date"] as! String
      var messageArray = message.componentsSeparatedByString(":")
letmessageType: String = messageArray[0];
letmessageContent: String? = messageArray[1];
ifsenderNickname == nickname{
cell.lblChatMessage.textAlignment = NSTextAlignment.Right
cell.lblMessageDetails.textAlignment = NSTextAlignment.Right
cell.imgChatMessage.contentMode = .TopRight
cell.lblChatMessage.textColor = lblNewsBanner.backgroundColor
}
if(messageType=="txt"){
cell.lblChatMessage.hidden=false;
cell.imgChatMessage.hidden=true;

cell.lblChatMessage.text = messageContent
cell.lblChatMessage.textColor = UIColor.darkGrayColor()
}else{
cell.lblChatMessage.hidden=true;
cell.imgChatMessage.hidden=false;

cell.imgChatMessage.image =
deltaDecompressImage(messageContent!,inTargetWidth:
CGFloat(Int(messageArray[2])!), inTargetHeight: CGFloat(Int(messageArray[3])!))
}
cell.lblMessageDetails.text = "by \(senderNickname.uppercaseString) @
\(messageDate)"

returncell
}
```

deltaDecompressImage metodu delta algoritmasıyla sıkıştırılan string haldeki resmi normal görsel öğeye geri çeviren metodudur.

```
funcdeltaDecompressImage(inStringImage:String,inTargetWidth:CGFloat,
inTargetHeight:CGFloat)->UIImage{
let size = CGSize(width: inTargetWidth, height: inTargetHeight)
letactualWidth:Int = Int(size.width)
letactualHeight:Int = Int(size.height)
letrect = CGRect(origin: CGPoint(x: 0,y: 0), size: size)
UIGraphicsBeginImageContextWithOptions(size, false, 1)
UIColor.whiteColor().setFill() // orcustomcolor
UIRectFill(rect)
varimage = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()
      var inArray:Array<Array<Int>> = [[0,0,0]]
letinStringImagePixels:Array<String>
inStringImagePixels =
(inStringImage.substringFromIndex(inStringImage.startIndex.advancedBy(1))).componentsSe
paratedByString("*")
for i in 0...inStringImagePixels.count-1 {
         var inStringImagePixelValues:Array<Int> = [0,0,0]
         var tempStringImagePixelValues:Array<String> =
inStringImagePixels[i].componentsSeparatedByString("|")
inStringImagePixelValues[0] = Int(tempStringImagePixelValues[0])!
inStringImagePixelValues[1] = Int(tempStringImagePixelValues[1])!
inStringImagePixelValues[2] = Int(tempStringImagePixelValues[2])!
inArray.append(inStringImagePixelValues)
}
inArray.removeAtIndex(0)
for y in 0...actualHeight-1 {
for x in 0...actualWidth-1{
letindex = y*actualWidth+x
            var currentRed:Int = 0
            var currentGreen:Int = 0
            var currentBlue:Int = 0
if(index>0){
currentRed = inArray[index-1][0]+inArray[index][0]
currentGreen = inArray[index-1][1]+inArray[index][1]
currentBlue = inArray[index-1][2]+inArray[index][2]
}else{
currentRed=inArray[index][0]
currentGreen=inArray[index][1]
currentBlue=inArray[index][2]
}
image = image!.setPixelColorAtPoint(CGPoint(x: x,y: y), color:
UIImage.RawColorType(UInt8(currentRed),UInt8(currentGreen),UInt8(currentBlue) ,255))!
}
}returnimage! }
```

processKNNinput olarak sıkıştırılacak resmi ham haliyle alan ve küme sayısını alan sonucunda KNN algoritmasıyla bize sıkıştırılmış resim döndüren metodudur.

```
funcprocessKNN(inUIImage:UIImage!,inNumberOfCluster:Int)->UIImage!{
letactualWidth = Int(inUIImage.size.width);
letactualHeight = Int(inUIImage.size.height);
letnumberOfCluster:Int = inNumberOfCluster
varpoints = Array<KNNClusterPoint>()

for y in 0...actualHeight-1 {
for x in 0...actualWidth-1 {
letindex = y*actualWidth+x;
letxLocation:Int = x
letyLocation:Int = y
letxx:CGFloat = CGFloat.init(xLocation)
letyy:CGFloat = CGFloat.init(yLocation)
letmyColor:UIColor? = inUIImage.getPixelColorAtLocation(CGPointMake(xx, yy))

letpoint = KNNClusterPoint.init(inXLocation: xLocation,inYLocation: yLocation,inColor: myColor,
inPointIndex: index)
points.append(point)
}
}
letrect = CGRect(origin: CGPoint(x: 0,y: 0), size: inUIImage.size)
UIGraphicsBeginImageContextWithOptions(inUIImage.size, false, 1)
UIColor.blackColor().setFill()
UIRectFill(rect)
varimage = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

varcentroids = Array<KNNClusterCentroid>()
for i in 0...numberOfCluster-1 {
letrandomFirst:CGFloat =
CGFloat(arc4random_uniform(arc4random_uniform(UInt32(actualWidth))));
letrandomSecond:CGFloat =
CGFloat(arc4random_uniform(arc4random_uniform(UInt32(actualHeight))));
letmyColor:UIColor? = inUIImage.getPixelColorAtLocation(CGPointMake(randomFirst,
randomSecond))

letcentroid = KNNClusterCentroid.init(inXLocation: Int(randomFirst),inYLocation:
Int(randomSecond),inColor: myColor)
centroids.append(centroid)
}
letnewKNN = KNN.init()
newKNN.knnExecute(points, inClusters: centroids, inMyImage: inUIImage, inNumberOfCluster:
inNumberOfCluster)
image = newKNN.calculateProcessedImage(image);
returnimage
}
```

processKMeansinput olarak sıkıştırılacak resmi ham haliyle alan ve küme sayısını alan sonucunda KNN algoritmasıyla bize sıkıştırılmış resim döndüren metodudur.

```
funcprocessKMeans(inUIImage:UIImage!,inNumberOfCluster:Int)->UIImage!{
letactualWidth = Int(inUIImage.size.width);
letactualHeight = Int(inUIImage.size.height);

letnumberOfCluster:Int = inNumberOfCluster

varpoints = Array<KMeansClusterPoint>()

for y in 0...actualHeight-1 {
for x in 0...actualWidth-1 {
letindex = y*actualWidth+x;
letxLocation:Int = x
letyLocation:Int = y

letxx:CGFloat = CGFloat.init(xLocation)
letyy:CGFloat = CGFloat.init(yLocation)

letmyColor:UIColor? = inUIImage.getPixelColorAtLocation(CGPointMake(xx, yy))

letpoint = KMeansClusterPoint.init(inXLocation: xLocation,inYLocation: yLocation,inColor:
myColor,inPointIndex: index)
points.append(point)
}
}
letrect = CGRect(origin: CGPoint(x: 0,y: 0), size: inUIImage.size)
UIGraphicsBeginImageContextWithOptions(inUIImage.size, false, 1)
UIColor.blackColor().setFill()
UIRectFill(rect)
varimage = UIGraphicsGetImageFromCurrentImageContext()
UIGraphicsEndImageContext()

varcentroids = Array<KMeansClusterCentroid>()
for i in 0...numberOfCluster-1 {
letrandomFirst:CGFloat = CGFloat(arc4random_uniform(UInt32(actualWidth)));
letrandomSecond:CGFloat = CGFloat(arc4random_uniform(UInt32(actualHeight)));

letmyColor:UIColor? = inUIImage.getPixelColorAtLocation(CGPointMake(randomFirst, randomSecond))

letcentroid = KMeansClusterCentroid.init(inXLocation: Int(randomFirst),inYLocation:
Int(randomSecond),inColor: myColor)
centroids.append(centroid)
}

letnewKMeans = KMeans.init()
newKMeans.KMeansExecute(points, inClusters: centroids, inMyImage: inUIImage, inNumberOfCluster:
inNumberOfCluster)

image = newKMeans.calculateProcessedImage(image);
returnimage
}
```

calculateEuclideanDistancemethoduKMeans ve KNN algoritmalarında noktalar ile küme merkezleri arasındaki uzaklığı ölçmek için yapılmıştır.

```
funccalculateEuclideanDistance( inCentroid:KNNClusterCentroid ,
inPoint:KNNClusterPoint ) ->Double {
letpointRed:Double
letpointGreen:Double
letpointBlue:Double

letcentroidRed:Double
letcentroidGreen:Double
letcentroidBlue:Double

      var myCIColor = inPoint.pixelColor!.coreImageColor
pointRed = Double(myCIColor!.red)
pointGreen = Double(myCIColor!.green)
pointBlue = Double(myCIColor!.blue)

myCIColor = inCentroid.pixelColor!.coreImageColor
centroidRed = Double(myCIColor!.red)
centroidGreen = Double(myCIColor!.green)
centroidBlue = Double(myCIColor!.blue)

letreValue = sqrt(pow(pointRed - centroidRed, 2) + pow(pointGreen -
centroidGreen, 2) + pow(pointBlue - centroidBlue, 2))
returnreValue
}
```

calculateProcessedImagemethoduKMeans ve KNN algoritmalarında kaynak resmin noktalarının renklerini kümeledikten sonra bu noktaları pixele çeviren methoddur.

```
funccalculateProcessedImage(inImage:UIImage!) ->UIImage! {
    var image:UIImage! = inImage

    //letrgba:RGBA = RGBA(image: image)

letexamplePixel:PixelData = PixelData(a: 0, r: 0, g: 0, b: 0)
    var pixels = [PixelData](count: Int(inImage.size.width) *
Int(inImage.size.height), repeatedValue: examplePixel)

for j in 0...points.count-1 {
let p:KNNClusterPoint = points[j]

letmyCIColor = p.pixelColor!.coreImageColor

pixels[j]=PixelData(a: UInt8((myCIColor?.alpha)!*255), r:
UInt8((myCIColor?.red)!*255), g: UInt8((myCIColor?.green)!*255), b:
UInt8((myCIColor?.blue)!*255))
}

image = imageFromBitmap(pixels, inWidth: Int(inImage.size.width), inHeight:
Int(inImage.size.height))
returnimage
}
```

ÖZGEÇMİŞ

Adı Soyadı : Ömer GÜNGÖR

Doğum Yeri ve Yılı : Karşıyaka, 1991

Medeni Hali : Evli

Yabancı Dil : İngilizce

E-Posta : gungoromer@windowslive.com

ÖĞRENİM DURUMU

Lise : Karşıyaka Anadolu Teknik Lisesi, Bilgisayar Yazılım Teknolojileri, 2005

Ön Lisans : Dokuz Eylül Üniversitesi, İzmir Meslek Yüksekokulu, Bilgisayar Programcılığı, 2009

Lisans : Anadolu Üniversitesi, Açıköğretim Fakültesi, İşletme Bölümü, 2011

Yüksek Lisans : İzmir Katip Çelebi Üniversitesi, Fen Bilimleri Enstitüsü, Sistem Mühendisliği Bölümü, 2014

MESLEKİ DENEYİM VE ÖDÜLLER

Şubat 2012 – Mayıs 2013 arasında Integer Yazılım firmasında yazılım geliştirici olarak çalıştım.

Temmuz 2013 – Ocak 2016 arasında Gediz Üniversitesi Bilgi İşlem Müdürlüğü'nde Yazılım Geliştirici olarak çalıştım. Özel olarak Üniversite Bilgi Sistemi projesi için görev aldım.

Ocak 2016 – Ekim 2016 arasında Sertom Yazılım firmasında Yazılım Geliştirme ve Proje Sorumlusu olarak çalıştım.

Ekim 2013 – Günümüze kadar Darkese.com E-Ticaret firmasında kurucu ortak ve yazılım geliştiricisiyim.

YÜKSEK LİSANS TEZİNDEN TÜRETİLEN YAYINLAR, SUNUMLAR VE PATENTLER

- Doc.Dr. Aysegul ALAYBEYOGLU, Omer GUNGOR "Image Compression And Transmission Applications With Artificial Intelligence Method For Ios Based Mobile Devices" (Hazırlık Aşamasında)

DİĞER YAYINLAR, SUNUMLAR VE PATENTLER

Ulusal Yayın

- Süleyman Emir TURNA, **Ömer GÜNGÖR**, "1 Taşla 3 Kuş: Active Directory, Üniversite Bilgi Sistemi ve İnternet Erişimi Entegrasyonu", Akademik Bilişim 2015, 2015