GERİ YAYILIMLI BİRLİKTE EVRİM İLE İYİLEŞTİRİLMİŞ DERİN SİNİR AĞLARI KULLANILARAK YOL ÇATLAK TESPİTİ

Emirhan Mustafa ANIK

T.C.
BURSA ULUDAĞ ÜNİVERSİTESİ
FEN BİLİMLERİ ENSTİTÜSÜ

GERİ YAYILIMLI BİRLİKTE EVRİM İLE İYİLEŞTİRİLMİŞ DERİN SİNİR AĞLARI KULLANILARAK YOL ÇATLAK TESPİTİ

Emirhan Mustafa ANIK
502026006

Prof. Dr. Turan ARSLAN
(Danışman)

YÜKSEK LİSANS
İNŞAAT MÜHENDİSLİĞİ ANABİLİM DALI

BURSA – 2022
Her Hakkı Saklıdır

TEZ ONAYI

ÖZET

Yüksek Lisans

GERİ YAYILIMLI BİRLİKTE EVRİM İLE İYİLEŞTİRİLMİŞ DERİN SİNİR AĞLARI KULLANILARAK YOL ÇATLAK TESPİTİ

Emirhan Mustafa ANIK

Bursa Uludağ Üniversitesi
Fen Bilimleri Enstitüsü
İnşaat Mühendisliği Anabilim Dalı

Danışman: Prof. Dr. Turan ARSLAN

Çatlaklar trafik yükleri etkisi ile alt tabakadan başlayarak kaplama yüzeyine kadar oluşan yüzeysel hasarlardır. Oluşan bir çatlak hasarının büyümeden tespit edilip gerekli bakımlarının yapılması hem yol konforuna hem de bakım için yapılacak harcamalara olumlu olarak katkıda bulunmaktadır. Bu çalışmada yol üzerinde bulunan çatlakları gerçek zamanlı ve yüksek doğrulukta tespit etmek amaçlanmıştır. Bu kapsamda Geri Yayımlı Birlikte Evrim yaklaşımıyla İyileştirilmiş Derin Sinir Ağları ve görüntü işleme yöntemlerini birlikte kullanılmıştır. Çalışmada çeşitli sayıda ve boyutta çatlak görsel verileri içeren EdmCrack600, AsphaltCrack, CFD ve CrackSegmentation veri setlerinden yararlanılarak yeni bir veri seti elde edilmiş ve bu veri seti üzerinde Derin Sinir Ağları tabanlı öğrenim gerçekleştirilmiştir. Görüntü işleme teknikleri sayesinde ise çatlak tespiti yapılan görsel içerisinden çatlak olmayan nesneler arındırılmış ve çatlağın kabaca konumunu gösteren siyah-beyaz bir resim elde edilmiştir. Son olarak kabaca konumu belirlenmiş çatlak üzerinde en iyi öğrenme gerçekleştirmiş ağ yapısına ait parametreler kullanılarak alan bazlı çatlak tespiti yapılmıştır. Modelin doğruluğu CFD veri seti kullanarak Keskinlik, Duyarlılık ve F1-Score kriterleri ile değerlendirilmiştir. Değerlendirme sonucunda, önerilen yöntem maksimum saniyede 48 görsel üzerinde çatlak tespiti yapabilirken %92,74 Kesinlik, %88,92 Duyarlılık ve %89,61 F1-Score başarı yüzdelerine erişebildiği gözlenmiştir.

Anahtar Kelimeler: Geri yayılımlı birlikte evrim, derin sinir ağları, çatlak tespiti, görüntü işleme

2022, vii + 94 sayfa.

ABSTRACT

MSc Thesis

ROAD CRACK DETECTION USING DEEP NEURAL NETWORKS DEVELOPED VIA COOPERATIVE COEVOLUTION WITH BACKPROPAGATION

Emirhan Mustafa ANIK

Bursa Uludağ University
Graduate School of Natural and Applied Sciences
Department of Civil Engineering

Supervisor: Prof. Dr. Turan ARSLAN

Cracks are superficial damages that occur from the substrate to the pavement surface due to the effect of traffic loads. Detecting a crack damage before it grows and performing the necessary maintenance contributes positively to both the road comfort and the expenses to be made for maintenance. In this study, it is aimed to detect the cracks on the road in real time and with high accuracy. In this context, Deep Neural Networks Developed via Cooperative Coevolution with Backpropagation and image processing and image processing methods were used together. In the study, a new data set was obtained by using EdmCrack600, AsphaltCrack, CFD and CrackSegmentation datasets containing cracked visual data in various numbers and resolutions, and Deep Neural Networks-based learning was performed on this dataset. Thanks to image processing techniques, objects without cracks were removed from the image in which cracks were detected, and a black-and-white picture showing the rough location of the crack was obtained. Finally, area-based crack detection was performed by using the parameters of the network structure that performed the best learning on the roughly positioned crack. The accuracy of the model was evaluated with Precision, Sensitivity and F1-Score criteria using CFD dataset. As a result of the evaluation,it has been observed that the proposed method can detect cracks on 48 images per second, while it can reach 92.74% Precision, 88.92% Recall and 89.61% F1-Score success rates.

Key words: Cooperative coevolution with backpropagation, deep neural networks, crack detection, image processing

2022, vii + 94 pages.

TEŞEKKÜR

Tez çalışmam sırasında kıymetli bilgi, birikim ve tecrübelerini benden esirgemeyen ve destek olan değerli danışman hocam Sayın Prof. Dr. Turan ARSLAN'a sonsuz teşekkürlerimi sunarım.

Hayatım her anında sevgi ve ilgilerini eksik etmeden her zaman maddi ve manevi olarak yanımda olan ve tez yazım sürecindeki desteklerinden ötürü aileme en içten şükran ve minnetlerimi sunarım.

Emirhan Mustafa ANIK
12/08/2022

İÇİNDEKİLER

Sayfa

ÖZET....vi
ABSTRACT....vii
TEŞEKKÜR....viii
SİMGELER VE KISALTMALAR DİZİNİ....x
ŞEKİLLER DİZİNİ....xi
ÇİZELGELER DİZİNİ....xiii
1. GİRİŞ....1
2. KAYNAK ARAŞTIRMASI....5
3. MATERYAL ve YÖNTEM....14
3.1. Yapay Sinir Ağları....14
3.1.1. İleri Beslemeli Yapay Sinir Ağı Modeli....16
3.1.2. Geri Beslemeli Yapay Sinir Ağı Modeli....17
3.2. Derin Sinir Ağları....18
3.2.1. Çok Katmanlı Algılayıcılar....19
3.3. Önerilen DSA Tabanlı Çatlak Tespit Yöntemi....20
3.3.1. Veri Seti....23
3.3.2. Geri Yayılımlı Birlikte Evrim ile Öğrenme Süreci....27
3.3.3. Geri Yayılımlı Birlikte Evrim Yöntemi....32
3.3.3.1. Geri Yayılım....35
3.3.3.2. Birlikte Evrim....36
3.3.3.2.1. Ayrıştırma....37
3.3.3.2.2. Seçim Stratejisi....38
3.3.3.2.3. Optimizasyon Süreci....39
3.3.3.3. Geri Yayılımlı Birlikte Evrim Algoritması....40
3.3.3.4. Kesinlik, Duyarlılık ve F1-Score....42
3.3.4. Görüntü İşleme ile Çatlak Bölgesinin Kabaca Belirlenmesi....43
3.3.4.1. Uyarlanabilir Eşikleme Yönteminin Uygulanması....43
3.3.4.2. Çok Küçük Nesne ve Boşlukların Giderilmesi....44
3.3.4.3. Küçük Boyutlu Nesnelerin Arındırılması....46
3.3.4.4. Nesnelerin Birleştirilmesi....51
3.3.4.5. Büyük Nesnelerin Temizlenmesi....53
3.3.5. Görsel İçerisinde Çatlak Alanının Tespit Edilmesi....55
3.3.5.1. Tespit Edilen Çatlak Bölgesinin Ana Resimde Göstermek....57
3.3.5.2. Önerilen Yöntemin Başarısının Değerlendirilmesi....57
4. BULGULAR ve TARTIŞMA....59
4.1. GYBE_DSA ile Optimum Öğrenme Performansı....59
4.2. Önerilen Yöntem ile Çatlak Tespitinin Yapılması....60
5. SONUÇ....65
KAYNAKLAR....67
EKLER....71
ÖZGEÇMİŞ....94

SİMGELER VE KISALTMALAR DİZİNİ

Simgeler	Açıklama
b	Yanlılık parametresi
β	Düzenlileştirme terimi
D	Eğitim verisi
NP	Popülasyon boyutu
px	Piksel
V	Tespit yapılan görselin alanı
w	Ağırlık parametresi
x	Girdi değeri
y	Çıktı değeri

Kısaltmalar	Açıklama
BE	Birlikte Evrim
ÇKA	Çok Katmanlı Algılayıcılar
DSA	Derin Sinir Ağları
DVM	Destek Vektör Makineleri
EA	Evrimsel Algoritma
FN	Yanlış Negatif
FP	Yanlış Pozitif
FPS	Frame Per Second (Saniyedeki Kare Sayısı)
GY	Geri Yayılım
GY_DSA	Geri Yayılımlı Derin Sinir Ağları
GYBE_DSA	Geri Yayılımlı Birlikte Evrim ile İyileştirilmiş Derin Sinir Ağları
KUDE	Kendi Kendine Uyarlanabilir Diferansiyel Evrim
MDÖ	Minimum Dikdörtgen Örtü
TP	Doğru Pozitif
YSA	Yapay Sinir Ağları

ŞEKİLLER DİZİNİ

Sayfa

Şekil 1.1.	Önerilen yöntemin aşamalarını gösteren akış şeması	2
Şekil 3.1.	Temsili yapay sinir ağı	15
Şekil 3.2.	Genellikle kullanılan aktivasyon fonksiyonları …….	15
Şekil 3.3.	Basit bir ağ yapısının öğrenim süreci	16
Şekil 3.4.	İleri beslemeli yapay sinir ağı	17
Şekil 3.5.	Geri beslemeli yapay sinir ağı	17
Şekil 3.6.	Geri yayımlı bir ağda ağırlık parametrelerinin güncellenmesi ….	18
Şekil 3.7.	Çok Katmanlı Algılayıcılar yönteminin temsili bir ağ yapısı …..	20
Şekil 3.8.	Önerilen yöntemdeki birinci sürecinin akış şeması ……...	21
Şekil 3.9.	Önerilen yöntemdeki ikinci sürecinin akış şeması ……	22
Şekil 3.10.	Önerilen yöntemdeki son aşamanın akış şeması ……...	23
Şekil 3.11.	Veri setlerinde yer alan görsellerden 3 farklı boyutta görsel çıkarılması…….	25
Şekil 3.12.	Rastgele olarak alınmış çatlak içeren ve içermeyen 80x80 piksel boyutunda görseller……	27
Şekil 3.13.	Renkli görseli gri renge çevirmek için yazılım kodları …….	28
Şekil 3.14.	Renkli bir görselin üç farklı renk uzayında görüntüsü ve piksel değerleri…….	29
Şekil 3.15.	Renkli bir görselin gri tona dönüştürülmesi ve piksel değerleri ...	30
Şekil 3.16.	Gri tonda bir resme Uyarlanabilir Eşikleme Yönteminin uygulanması ……...	31
Şekil 3.17.	Rastgele olarak seçilmiş 80x80'lik çatlak içeren ve içermeyen görsellere eşikleme yönteminin uygulanması sonucu elde edilen görseller……...	32
Şekil 3.18.	Hata miktarı ve arama uzayına bağlı olarak yerel ve küresel optimumum temsili gösterimi…….	33
Şekil 3.19.	Birlikte Evrim yönteminin akış diyagramı …….	36
Şekil 3.20.	Sinir ağlarının alt görevlere ayrıştırılmasının görünümü ……	37
Şekil 3.21.	GYBE_DSA yönteminin algoritmasında kullanılan simgeler ve açıklamaları …….....	41
Şekil 3.22.	GYBE_DSA yönteminin algoritması …….	41
Şekil 3.23.	Görselin gri renge çevrilip eşikleme yönteminin uygulanması …..	44
Şekil 3.24.	Görseldeki çatlak bölgesinin içinde yer alan çok küçük boşluk ….	45
Şekil 3.25.	Görsel içerisindeki çok küçük nesne ve boşluklardan kurtulmak için "scikitimage" kütüphanesinin kullanılması……..	45
Şekil 3.26.	Görselde bulunan nesnelerin dik açılı dikdörtgen alan yöntemi uygulanarak dikdörtgenler ile temsil edilmesi…….	47
Şekil 3.27.	Görselde bulunan nesnelerin en küçük dikdörtgen alan yöntemi uygulanarak dikdörtgenler ile temsil edilmesi ……	48
Şekil 3.28.	Görsel içerisinde bulunan küçük boyutlu nesnelerin görsel içerisinden silinmesini sağlayan akış diyagramı……..	50
Şekil 3.29.	Görsel içerisinde bulunan belirli boyuttaki çatlak olmayan nesnelerin silinmesi……..	51
Şekil 3.30.	Görselde bulunan her bir nesnenin alınabilecek uç noktalarından bölgeler alınması:……...	52

Şekil 3.31. Görselde bulunan ve birbirine yakın olan iki parçaya ayrılmış çatlak nesnelerinin birleştirilmesi…….. 52
Şekil 3.32. Görsel içerisinde bulunan büyük boyutlu nesnelerin görsel içerisinden temizlenmesini sağlayan akış diyagramı……... 54
Şekil 3.33. Görselde bulunan büyük çatlak olmayan nesnelerin temizlenmesi .. 54
Şekil 3.34. Çatlak tespiti yapılacak görselde çatlak bölgelerin belirlenmesi….. 56
Şekil 3.35. Çatlak bölgeleri belirlenmiş alanlarda çatlak tespitinin yapılması… 56
Şekil 3.36. Çatlak bölgesinin orijinal resim üzerinde gösterilmesi …… 57
Şekil 3.37. Çatlak tespiti için olması gereken ve önerilen yöntem ile bulunan çatlak alanları…… 58
Şekil 4.1. Çatlak tespiti yapılan görsellerinin sırasıyla orijinal, çatlak olmayan nesnelerden arındırılmış ve çatlak tespit edilen bölgeleri belirtilmiş halleri……... 61

ÇİZELGELER DİZİNİ

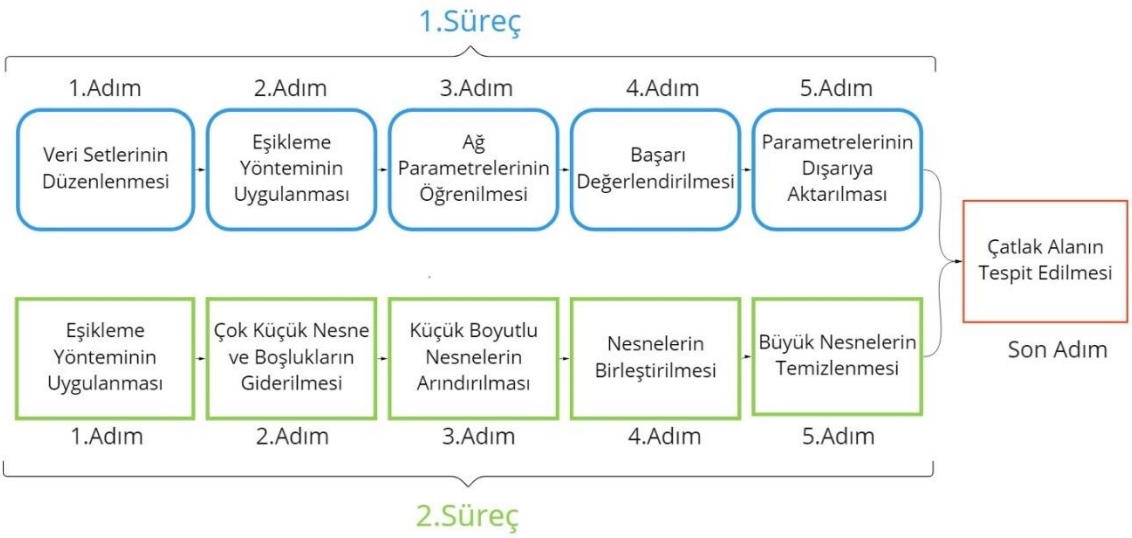
Sayfa

Çizelge 3.1. Çalışmada kullanılan veri setlerinin örnek sayısı ve görsel boyutu..... 24
Çizelge 3.2. Belirlenen boyutlara bağlı olarak gereken minimum örnek sayısı..... 26
Çizelge 3.3. Dik açılı dikdörtgen alan yöntemi uygulanmasıyla öğrenilen nesnelerin geometrik özellikleri..... 47
Çizelge 3.4. En küçük dikdörtgen alan yöntemi uygulanmasıyla öğrenilen nesnelerin geometrik özellikleri..... 49
Çizelge 4.1. Çeşitli ağ yapılarına göre öğrenme başarısının 3 farklı yöntem ile değerlendirilmesi..... 59
Çizelge 4.2. CFD veri seti kullanarak çatlak tespiti yapan yöntemler ve tespit başarıları..... 61
Çizelge 4.3. Çatlak tespiti yapılan bilgisayarların teknik özellikleri 63
Çizelge 4.4. Çatlak tespiti yapan cihazların FBS değerleri 63
Çizelge 4.5. Çatlak tespiti açısından bazı yöntemlerin başarısı ve FBS değerleri..... 63

1. GİRİŞ

Günümüzde teknolojinin gelişmesiyle beraber mevcut sistemler gelişirken yeni sistemlerde ihtiyaçlara göre ortaya çıkmaktadır. Bu sistemlerden biri olan Akıllı Ulaşım Sistemleri insan hayatında çok büyük bir öneme sahiptir ve bu nedenle yıllardır bu sistem üzerine çeşitli çalışmalar yapılmaktadır. Yapılan çalışmalar incelendiğinde genel olarak trafik ve ulaşımın en temel parçası olan yol yapılarına odaklanıldığı ve ulaşım sistem elemanlarının incelenerek geliştirilmesi için büyük bir emek harcandığı görülmektedir. Akıllı Ulaşım Sistemleri alanında yapılan çalışmaların artmasıyla beraber yol yapıları üzerinde daha detaylı ve çeşitli çalışmalar yapılmaya başlanmış ve özellikle yol üzerinde bulunan çatlakların tespit edilmesine yönelik çalışmalara önem verilmiştir. Çatlak tespiti ile ilgili çalışmalar incelendiğinde Derin Sinir Ağları (DSA) yaklaşımının en popüler sistem olduğu ve bu sistem temel alınarak çeşitli çalışmalar önerildiği göze çarpmaktadır. Bu çalışmada da Bölüm 3'te açıklanan DSA tekniklerinden biri olan Çok Katmanlı Algılayıcılar (ÇKA) modeli ile çeşitli görüntü işleme metotları kullanılarak çatlak tespiti için yeni bir yöntem önerilmiştir.

Bu çalışmada izlenen yöntemler Şekil 1.1'de verilen akış diyagramında gösterilmiştir. Ayrıca önerilen çatlak tespiti yöntemi için PHYTON yazılımı ve yazılımın içerisinde yer alan çeşitli kütüphanelerden yararlanılmıştır. Temel olarak yöntem Bölüm 3'te detaylı bir şekilde açıklanacak iki süreçten oluşturulmuş ve 1.Süreç'te istenilen parametre verilerini elde ettikten sonra 2.Süreç'e başlanmıştır. Bu nedenle ilk başta 1.Süreç açıklanmıştır. Bu sürecin 1.Adım'ında literatürde kullanılan dört farklı çatlak veri setinde yer alan farklı boyutlardaki görsellerden 80 x 80 piksel boyutlarında çatlak içeren ve içermeyen görseller alınmıştır. Bu boyutta bölgelerin alınmasında görsel içerisindeki çatlakların boyutu ve elde edilecek örnek sayısı olmak üzere iki temel faktör etkili olmuştur ve bu nedenler Bölüm 3'te detaylıca açıklanmıştır.



Şekil 1.1. Önerilen yöntemin aşamalarını gösteren akış şeması

Veri setlerindeki görsellerden 80 x 80 piksel boyutlarında görseller elde edildikten sonra 2.Adım'a geçilmiştir. Bu adımda Bölüm 3'te detaylandırılmış Uyarlanabilir Eşikleme Yöntemi uygulanmıştır. Eşikleme yöntemi ile her pikselde 255 farklı değer alan gri renkteki bir görsel sadece 0 ve 255 değeri alan siyah beyaz bir görsele çevrilmiş. Böylece işlenen verinin boyutunu azaltılmış ve zaman açısından kazanç sağlanmıştır. Sonrasında 3.Adım'a geçilmiş ve Geri Yayılımlı Birlikte Evrim Yaklaşımıyla İyileştirilmiş Derin Sinir Ağları yöntemi (GYBE_DSA) (Gong, Liu, Qin, Zhao ve Tan, 2021) kullanılarak belirtilen boyuttaki bölge içerisinde çatlak olup olmadığını tespit eden ve ağ parametre değerlerini içeren bir çıktı verisi elde edilmiştir. Ardından 4.Adım'da elde edilen bu çıktı verilerinin başarılıları; pozitif olarak tahminlenen değerlerin gerçekte kaç adedinin pozitif olduğunu gösteren Kesinlik, pozitif olarak tahminlenen değerlerin gerçekte kaç adedinin pozitif olarak tahmin edildiğini gösteren Duyarlılık ve bu iki yöntem ile elde edilen değerlerin harmonik ortalamasını gösteren F1-Score yöntemleri ile değerlendirilmiştir. Bu yöntemler bir sınıflama modelinde gerçek değerler ile tahmin değerleri karşılaştırarak bu model içerisinde yapılan tahminlerin ne ölçüde başarılı olduğunu göstermekte ve yöntemler Bölüm 3'te daha detaylı bir şekilde açıklanmaktadır. Değerlendirme sonrasındaki 5.Adımda en yüksek başarıya sahip parametre verileri daha sonra kullanılmak üzere pickle formatında dışarıya aktarılmıştır. Pickle ise PHYTON yazılımı içerisinde yer alan özel bir depolama modülüdür. Bu modül sayesinde çeşitli veri türleri depolanıp dışarıya aktarılabilir ve daha sonra içeriye aktarılarak tekrar kullanılabilir.

Dışarıya aktarılan veriler kullanılarak 80 x 80 piksel boyundaki bir bölgede çatlak tespiti yapılabilmektedir. Ancak normal bir görsel içerisinde bulunan çatlağın görseldeki konumu bilinmediği için hangi bölgede tespit yapılacağı da bilinmemektedir. Bu nedenle görsel içerisinde yer alan çatlakların konumunun kabaca bilinmesi taranacak bölgelerin bilinmesini sağlayacak ve zaman açısından önemli bir katkıda bulunacaktır. Dolayısıyla çatlakların konumunun tahmini olarak belirlenmesi için 2.Süreç'te yer alan çeşitli görüntü işleme yöntemleri içeren 5 farklı adım uygulanmıştır.

1.Adım'da çatlak tespiti yapılmak istenen görsele Uyarlanabilir Eşikleme Yöntemi uygulanmıştır. Ardından 2.Adım'a geçilmiş ve görsel içerisinde yer alan çok küçük boyutlarda çatlak olmayan nesneler görüntü işleme ilgili çeşitli algoritmalar içeren "scikit-image" kütüphanesinde yer alan yöntemler kullanılarak temizlenmiş ve nesnelerin içerisinde yer alan çok küçük boşluklar çatlak tespiti sırasında nesnelerin doğru algılanabilmesi için doldurulmuştur. Bu işlemin ardından 3.Adım'da çatlak içermemesine rağmen çatlak olarak algılanıp bu alanda tarama yapılmasına ve harcanan zamanı artırmasına neden olan küçük boyutta bulunan nesneler, veri setindeki görsellerde yer alan nesnelerin geometrik özelliklerinden yararlanarak görsel içerisinden çıkarılmıştır. Bu işleme takiben 4.Adım'da, birbirine oldukça yakın olan çatlak nesneleri birleştirilmiş ve böylece taranacak alan sayısı azaltılmıştır. Ardından 5.Adım'da görsel içerisinden yer alan büyük boyutlu çatlak içermeyen nesneler çeşitli geometrik özellik değerleri kullanılarak silinmiştir.

Son Adım'da da 2.Süreç sonucunda elde edilen görsel içerisinde yer alan bölge parçalanarak taranacak alanlar belirlenmiş ve belirlenen bölgelerde 1.Süreç'te GYBE_DSA tekniğiyle öğrenilen parametreler kullanılarak çatlak tespiti yapılmıştır. Çatlak olarak tespit edilen alanlar sistem içerisinde kaydedilmiş ve ana görsel üzerinde çatlak bölgesi belirlenmiştir. Ek olarak yöntemin başarısı Kesinlik, Duyarlılık ve F1-Score yöntemleri ve çatlak görsel verileri içeren CFD (Lyu ve diğerleri, 2019; Shi, Cui, Qi, Meng ve Chen, 2016) veri seti kullanılarak test edilmiştir. Kullanılan veri seti içerisinde yer alan ve GYBE_DSA yöntemi ile öğrenmede kullanılmayan 80 adet görsel veri kullanılmıştır. Burada CFD veri setinin kullanılmasında temel bir neden vardır. Önerilen yöntem özellikle herhangi bir aracın alt kısmına yerleştirilebilecek bir kamera

ile çatlak tespit yapan bir mekanizmada kullanılabilmesi için tasarlanmıştır ve CFD veri setindeki çoğu görsel bahsedilen mekanizmada elde edilecek görseller gibi dik açıdan çekilmiş ve yol dışında bulunan başka obje veya cisimleri içermeyen verilere sahiptir.

Tezin takip eden bölümlerinden "Kaynak Araştırması" olarak adlandırılan Bölüm 2'de DSA ve Görüntü İşleme hakkında kısa bir bilgi verilip yapılmış bazı çalışmalara yer verilmiştir. Bölüm 3 olan "Materyal ve Yöntem" bölümünde, kullanılan veri setleri, Çok Katmanlı Algılayıcılar tekniği, çeşitli görüntü işleme yöntemlerine ilişkin detaylı açıklamalar yapılmıştır. "Bulgular ve Tartışma" bölümünün yer aldığı Bölüm 4'te Kesinlik, Duyarlılık ve F1-Score yöntemleri kullanılarak çeşitli ağ yapılarının başarı sonuçlarına, çatlak tespit yönteminin uygulamalarına ve tespit süresine değinilmiş ve elde edilen değerler yorumlanmıştır. "Sonuç" olarak isimlendirilen Bölüm 5'te ise tezin genel olarak sonuçları özetlenmiş, önerilen yöntem ve uygulamalar üzerinde değerlendirmeler yapılmış ve konu üzerinde ileride yapılacak çalışmalarla ilgili bazı önerilerde bulunulmuştur.

2. KAYNAK ARAŞTIRMASI

Günümüze kadar yol çatlak tespiti için çeşitli yöntemler kullanılmıştır. Temelleri çok eskiye dayanan bu yöntemlerden biri de görüntü işleme uygulamalarıdır. Görüntü işlemenin birçok tekniği 1960 yıllarında Jet Propulsion Laboratuvarı, Massachusetts Teknoloji Enstitüsü gibi araştırma tesislerinde geliştirilmiştir (Solomon ve Breckon, 2010). Ancak o dönemde bilgisayar donanımları sayısal veri işleme için yeterince etkili olamamıştır. 1970 yıllarında bilgisayarların yaygınlaşmasıyla birlikte daha kolay bulunabilen ve daha verimli bir şekilde çalışabilen bilgisayar donanımları geliştirilmiştir. Artık gelişmiş donanımlar kullanılarak gerçek zamanlı olarak görüntü işleme uygulanabiliyordu ve bu durum sayısal görüntü işleme yöntemlerini yaygınlaştırmıştır. 2000'li yıllara gelindiğinde ise hızlı bilgisayarların ulaşılabilirliği sayısal olarak görüntü işlemeyi o dönemde kullanılan en yaygın yöntemlerden biri yapmıştır (Anonim, t.y.).

Görüntü işlemenin yaygınlaşması farklı alanlardaki araştırmacıların bu yöntemleri kullanmaya teşvik etmiştir. Yol üzerindeki çatlakların tespiti de bu alanlardan biridir ve 2000'li yıllarda bu alanda birçok çalışma yapılmıştır. Oliveire ve Correia (2009) üstyapı çatlaklarını belirlemek için dinamik eşikleme ve görüntü entropi teknolojisini birleştirerek morfolojiye dayalı olarak üstyapı görüntülerinin ön işlenmesini önermişlerdir. Tsai ve diğerleri (2010) farklı yol koşulları altında çeşitli yol görüntüleri için çatlak bölütleme yöntemlerini karşılaştırmıştır. Sonuç olarak dinamik programlamaya dayalı dinamik optimal eşik bölütleme yönteminin en iyi performansa sahip olduğunu gösterilmiştir ancak yüksek hesaplama karmaşıklığı ve hesaplama tüketimi nedeniyle pratik çalışmalarda sınırlı kalmaktadır. Bu yöntemin iyi bir performans gösterdiğini bildirmişler ve görüntü veri tabanındaki piksel yoğunluğu farkını daha da azaltmak için ek filtreleme teknolojisinin düşünülmesi gerektiğini önermişlerdir.

Na ve diğerleri (2011) küçük çatlakları tespit eden geleneksel algoritmadan daha etkili olan kesirli diferansiyel ve matematiksel morfolojiye dayalı bir çatlak tespit algoritması önermişlerdir. Önerilen yöntem yüksek verimliliğe sahip olmasına rağmen aydınlatma ve arka plan değişimlerine karşı etkili olamamıştır. Gavilan ve diğerleri (2011) bir dizi görüntü işleme tekniğini birleştiren bir yaklaşım önermişlerdir. İlk başta doğrusal

özellikleri geliştirmek için görüntüler ön işleme tabi tutulmuş ve kaplamalardaki derzler veya doldurulmuş çatlaklar gibi kafa karıştırıcı alanları ortadan kaldırmak için çatlak olmayan nesnelerin özellik tespiti yapılmıştır. Ardından çok yönlü minimum olmayan süprezyon tekniği ile simetri kontrolü metodunu birleştiren tohum tabanlı yeni bir yaklaşım önermişlerdir.

Zou ve diğerleri (2012) CrackTree adlı üç adımlı bir yöntem geliştirmiştir. Önerilen yöntemde, görsel içerisindeki gölge önce jeodezik tabanlı bir algoritma kullanılarak kaldırılmış ve daha sonra tensör oylamasına dayalı bir olasılık haritası oluşturulmuştur. Son olarak ise çatlakları belirlemek için olasılık haritasında oluşturulan minimum yayılan ağaç üzerinde özyinelemeli ağaç kenarı budaması yapılmıştır. Amhaz ve diğerleri (2016) ise çalışmalarında çatlakların kalınlığının tahmin edilebilmesi için rafine edilmiş bir artefakt filtreleme adımı ile geliştirilmiş bir minimal yol seçim algoritması tanıtmıştır.

Genel olarak, bu yöntemlerin en büyük avantajı eğitim sürecinin gerekli olmamasıdır ve bu nedenle yöntemleri uygulamak ve performansı doğrulamak daha kolaydır. Ayrıca bu yöntemler hesaplama açısından verimlidir. Bu tür yöntemlerin en büyük dezavantajı belirli veri kümeleri üzerinde bulunan özelliklerin çoğunun el ile girilmesi gerekmektedir ve aydınlatma değişiklikleri veya düzensiz çatlak şekli gibi gerçek hayattaki görüntülerdeki tüm varyasyonları göz önünde bulunduramazlar. Dolayısıyla kontrollü bir ortamda geliştirilen yöntemler durum değiştiğinde iyi performans gösteremezler.

Kaplama yüzeylerinin karmaşıklığı, aydınlatmadaki değişimi ve çatlakların şekillerindeki düzensizliği fark eden araştırmacılar, çatlak tespiti için makine öğrenimi tabanlı algoritmalar aramaya yönelmişlerdir (Shi ve diğerleri, 2016). Görüntü işleme yöntemlerine kıyasla makine öğrenimi tabanlı algoritmalar eğitim sürecinde çatlakların görünümünü etkileyebilecek çeşitli faktörleri dolaylı olarak dikkate alabilmektedir ancak Destek Vektör Makineleri (DVM) ve Yapay Sinir Ağları (YSA) (McCulloch ve Pitts, 1943) gibi geleneksel makine öğrenimi yöntemleri genellikle daha fazla hesaplama kaynağı gerektirmektedir. Aşağıda geleneksel makine öğrenimi tabanlı bazı çalışmalara yer verilmiştir.

Kaseko ve Ritchie (1993) geleneksel görüntü işleme teknikleri ile yapay sinir ağı modellerinin entegrasyonunu kullanarak otoyol kaplama video görüntülerinin işlenmesini otomatikleştirmek için bir metodoloji sunmuşlardır. Geliştirdikleri model video görüntülerinde tespit edilen çatlakların türü, şiddeti ve kapsamına göre kaplama yüzeyindeki çatlakları sınıflandırabilmiştir. Chou, O'Neill ve Cheng (1994) çalışmalarında kaplama görüntülerini analiz etmek için moment değişmezlerini ve sinir ağlarını uygulamaya yönelik yeni bir yaklaşım sunulmuşlardır. Farklı tehlike türlerinden moment değişmezleri hesaplanarak öznitelikler elde edilmiş ve ardından bu özellikleri sınıflandırmak için YSA kullanılmıştır. Lee ve Lee (2004) dijital kaplama görüntülerinin çatlak tiplerini sınıflandırmak için entegre bir sinir ağı tabanlı çatlak tespit sistemi sunmuşlardır. Bu sistem görüntü tabanlı sinir ağı, histogram tabanlı sinir ağı ve yakınlık tabanlı sinir ağı olmak üzere üç sinir ağı modeli içermektedir. Bu üç sinir ağı modeli, dijital kaplama görüntülerindeki çatlak piksellerinden ziyade alt görüntülere dayalı olarak çeşitli çatlak türlerini sınıflandırmak için geliştirilmiştir.

Li, Hou, Yang ve Dong (2009) çalışmalarında DVM parametrelerini optimize etmek için genetik algoritmanın kullanılmasına öncülük etmişlerdir. Parametreleri deneme yanılma yöntemiyle elde edilen DVM ve YSA performansları çatlak görüntüleri içeren veri seti kullanılarak karşılaştırılmıştır. Değerlendirme sonucunda DVM'nin YSA'dan daha iyi çalıştığı gözlenmiştir. Hu ve diğerleri (2010) kaplamayı doku yüzeyi ve çatlakları homojen olmama olarak ele almış ve özellikleri çıkarmak için doku analizi ve şekil tanımlayıcılarını kullanmışlardır. Bir alt bölgenin çatlak olup olmadığını sınıflandırmak içinse DVM modelini tercih etmişlerdir. Moussa ve Hussain (2013) altyapı elemanları üzerindeki yüzey hasarını tespit etmek ve sınıflandırmak ve bunları bir dizi yüzey hasarı görüntüsüne başarıyla uygulamak için yarı otomatik gelişmiş bir doku segmentasyonu yaklaşımı sunmuşlardır. Önerilen yaklaşım belirli bir renk aralığında piksel yoğunluğu değerleriyle ilgili ölçümleri içeren bir özellik vektörü tanımlayarak çeşitli renk uzaylarında uzamsal olarak komşu piksellerin istatistiksel analizini ve nicelleştirilmiş bir gri seviye ölçeğinde hesaplanan istatistikleri içermektedir. Özellik vektörünü sınıflandırmak için parametre optimize edilmiş doğrusal olmayan DVM kullanılmıştır.

Mathavan ve diğerleri (2015) çatlak görüntülerine kendi kendini organize eden denetimsiz bir öğrenme algoritması uygulamışlardır. Doku ve renk özellikleri, çatlakları arka plandan ayırt etmek için kendi kendini organize eden haritaya entegre edilmiştir. Shi ve diğerleri (2016) ise rastgele yapılandırılmış ormanlara dayalı bir çatlak tespit yöntemi önermişlerdir. Önerdikleri yöntemde, yapılandırılmış bilgi ile çatlak belirteçlerini öğrenmek için integral kanal özelliklerini kullanmışlardır. Ardından, belirteçlerini işlemek ve çatlakları bulmak için rastgele yapılandırılmış orman sistemini uygulamışlardır. Wang, Qiu, Wang, Xiao and Wang (2017) çatlama türlerini otomatik bir şekilde akıllı bir şekilde tanımlamak için DVM tabanlı bir yöntem geliştirmişlerdir. Üstyapı çatlakları Minimum Dikdörtgen Örtü (MDÖ) modeli kullanılarak gruplandırılmıştır. Bu çalışmada, MDÖ'nün göreceli konumu, oryantasyonu ve boyutunun yanı sıra, çatlama yoğunluğu ve çatlama bağlantısı gibi çatlama özellikleri kullanılarak üç farklı DVM modeli oluşturulmuş ve karşılaştırılmıştır.

Hoang (2018) asfalt kaplama yüzeyindeki hasarları tespit etmek için bir yapay zekâ modeli oluşturmuştur. Sayısal görüntülerden özniteliklerin çıkarılması için Gauss filtresi, yönlendirilebilir filtre ve entegre izdüşüm gibi görüntü işleme yöntemleri kullanılmıştır. En küçük kareler DVM ve YSA dahil olmak üzere iki makine öğrenme algoritmasını eğitmek ve doğrulamak için 200 görüntü örneğinden oluşan bir veri seti toplanmıştır.

Geleneksel makine öğrenimi tabanlı yöntemler, eğitim süreci nedeniyle görüntü işlemeli yöntemlerden daha iyi doğruluk elde edebilmesine rağmen, bu yöntemlerin girdileri hala araştırmacılar tarafından girilmesi gerekmektedir ve öğrenimin gerçekleşebilmesi için çok fazla kaynağa ihtiyaç duymaktadırlar. Ayrıca geleneksel makine öğrenimi yöntemlerinin kullanılması ile çok fazla girdi değerine sahip görseller üzerinde öğrenim gerçekleştirilmesi sonucunda istenilen performanslara erişilememektedir.

Ancak 2006 yılında makine öğreniminin en yenilikçi yöntemlerinden biri olan Derin Öğrenme (Hinton, Osindero ve Teh, 2006) sisteminin ortaya çıkması ile birlikte araştırmacıların odağı tamamen değişmiştir. Çok hızlı bir şekilde neredeyse tüm çalışmalara uyum sağlayan ve çok fazla katman veya nöron içermesi nedeniyle DSA olarak da adlandırılan Derin Öğrenme, 2016 yılında ilk kez yol çatlak tespiti

çalışmalarında kullanılmaya başlanmış ve bu sistem ile yapılacak çalışmaların önünü açmıştır. İçerdiği ağ yapısı sayesinde çok karmaşık problemleri geleneksel makine öğrenim yöntemlerine göre daha kısa sürede ve daha başarılı olarak çözebilmiştir. Aşağıda günümüze kadar DSA teknikleri kullanılarak çatlak tespiti yapan bazı çalışmalara yer verilmiştir.

Zhang, Yang, Daniel Yang ve Zhu (2016) çalışmasında ilk kez DSA tekniklerin biri olan Evrişimli Sinir Ağları yöntemini yol çatlak tespitinde uygulamış ve birçok çalışmaya öncelik etmişlerdir. Önerdikleri yöntemde görüntü kalitesi açısından çok iyi olmayan telefon gibi cihazlardan elde edilen çatlak görsellerini kullanmış ve çalışmanın yayımlandığı dönemdeki diğer yöntemlere göre oldukça büyük bir başarı göstermişlerdir.

Balaji, Thiru Balaji, DInesh, Nair ve Harish Ram (2018) çalışmalarında çatlak tespiti için 12 farklı model önermişlerdir. Altı tanesi Evrişimli Sinir Ağları tabanlı yöntemlerken, diğer altı tanesi ise görüntü işleme ve makine öğrenmesinin birleştirilerek tasarlanan modellerdir. Çalışmada yer alan 12 farklı model beş farklı sistem kullanarak modellerin performansları değerlendirilmiştir. Çalışmanın sonucunda derin öğrenme tabanlı modellerin çatlak tespiti için daha uygun olduğu gözlenmiştir.

Zhang ve diğerleri (2017) Evrişimli Sinir Ağları tabanlı ve verimli bir mimariye sahip olan CrackNet yöntemi kullanılarak 3 Boyutlu yol görselleri üzerinde çatlak tespiti yapabilen yeni bir yöntem önermişlerdir. Yaygın olarak kullanılan Evrişimli Sinir Ağlarından farklı olarak CrackNet, önceki katmanların çıktılarını küçülten havuzlama katmanına sahip değildir ve tüm katmanlarda yeni geliştirilmiş değişmez görüntü genişliği ve yüksekliği tekniğini kullanarak temelde piksel doğruluğunu mükemmel hale getirmeye çalışmaktadır. Önerilen yöntem, 1.800 adet 3D kaplama görüntüsü kullanılarak eğitilmiştir. Sunulan modelin başarısı 200 adet 3D kaplama görüntüsü üzerinde değerlendirilmiş ve %90,13 Kesinlik, %87,63 Duyarlılık ve %88,86 F1-Score başarı yüzdelerine erişebilmiştir.

Zhang ve diğerleri (2018) 3 Boyutlu yol görselleri üzerinde çatlak tespiti yapmak amacıyla CrackNet-R olarak adlandırdıkları Yinemeli Sinir Ağı tabanlı bir yöntemi

önermişlerdir. Ayrıca önerilen yöntem içerisinde Çok Katmanlı Algılayıcılar tekniğinden de yararlanmış ve çatlak tespiti için 3 boyutlu 3.000 adet çatlak görseli kullanılmıştır Önerilen modelin başarısı Kesinlik, Duyarlılık ve F1-Score yöntemleri ile değerlendirildiğinde sırasıyla %88,89, %95,00 ve %91,84 başarı değerlerine ulaşmışlardır. Ek olarak önerilen yöntem 2017 yılında Zhang ve diğerleri tarafından geliştirilen Evrişimli Sinir Ağları tabanlı CrackNet (A. Zhang ve diğerleri, 2017) algoritması ile karşılaştırılmış ve yukarıda bahsedilen üç başarı yöntemi açısından değerlendirildiğinde Yinemeli Sinir Ağı tabanlı CrackNet-R algoritmasının CrackNet algoritmasından daha başarılı olduğu gözlenmiştir.

Mandal, Uong ve Adu-Gyamfi (2019) yaptıkları çalışmada nesne tespiti yapmak için kullanılan ve Evrişimli Sinir Ağı yöntemi olan YOLO v2 (Redmon, Divvala, Girshick ve Farhadi, 2016) kullanılarak derin öğrenmeye dayalı otonom çatlak tespiti ve sınıflandırması için bir yöntem önermişlerdir. YOLO yönteminin açılımı “You Only Look Once” yani “Sadece Bir Kez Bak” olarak çevrilmektedir ve bu adın seçilmesinin temel nedeni ise algoritmanın sahip olduğu ağ yapısı sayesinde görsellerin içerdiği nesneleri tek seferde tespit edecek kadar hızlı olmasıdır. Günümüzde farklı ağ yapılarına sahip 10’dan fazla YOLO modeli bulunmaktadır. Çalışma içerisinde 7.240 eğitim ve 1.813 test yol görseli kullanmışlardır. Bu yöntem sayesinde acil bakım gerektiren yollardaki hasarlar belirlenebilmektedir. Ayrıca F1-score ile yöntemin başarı değerlendirmesinde çatlak tespitinde yaklaşık %88 başarıya ulaşırken çatlak sınıflandırmasında %74 başarı oranına erişilmiştir.

Zou ve diğerlerinin (2019) çalışmasında çatlak tespiti için kablosuz yüksek kaliteli kamera tabanlı bir sistem tanıtmışlardır. Sistem içerisinde bulunan kamera tespit yapacak polis arabası gibi bir aracın alt kısmına yerleştirilmektedir. Ardından kamera tarafından yakalanan görseller gerçek zamanlı ve kablosuz olarak çalışma istasyonuna çatlak tespiti için gönderilmektedir. Tespit için ise sığ ve derin öğrenme yöntemleri kullanılmış ve derin öğrenme yöntemlerinin sığ yöntemlere göre daha iyi performans gösterdiği gözlenmiştir.

Cao ve diğerleri (2019) sınıflandırıcı olarak üç farklı ayrımcı öğrenme tabanlı model, Çekişmeli Üretici Ağ öğrenme tabanlı metotlar ve klasik makine öğrenme sistemleri geliştirmişlerdir. Çalışmalarının sonucunda klasik DSA yöntemlerinden daha da derin ve karmaşık ağ yapısına sahip VGG-19 (Simonyan ve Zisserman, 2014) çatlak tespit yönteminin %89 F1-score başarı yüzdesiyle en başarılı sınıflandırıcı olduğu gözlenmiştir. VGG temelde bir Evrişimli Sinir Ağı modelidir ancak içerdiği karmaşık ağ yapısı nedeniyle Evrişimli Sinir Ağları yöntemlerinden farklı bir şekilde çalışmaktadır ve VGG-16, VGG-19 gibi çeşitli ağ yapılarına sahip VGG modelleri bulunmaktadır. Bu çalışma sayesinde ilk kez kablosuz olarak veri ileten kamera tabanlı bir sistem önerilmiş ve basit VGG modellerinden daha başarılı model sunulmuştur.

Lyu, Wang ve Wei (2019) yaptığı çalışmada DSA tabanlı otonom bir yöntem önermişlerdir. Çalışmada ilk başta çatlak tespiti yapılmak istenen görüntü işlenmekte ve ardından işlenmiş görüntü Evrişimli Sinir Ağları yönteminde kullanılmak üzere bir sonraki aşamaya geçmektedir. Duyarlılık değeri kullanılarak yapılan karşılaştırmalara göre yol çatlaklarının özelliklerini kullanarak tespit yapan bu metodun başarısı %83,5 başarı olasılığı ile doğru ve iyi bir yöntem olduğuna karar verilmiştir.

Hassan, Han ve Shin (2020) çalışmalarında çatlak tespiti için Evrişimli Sinir Ağları kullanarak geliştirilmiş YOLO modeli ve orijinal YOLO modeli kullanılmıştır. Her iki yöntem Doğruluk ve Ortalama Kesinlik Değerlerinin Ortalaması (OKDO) açısından karşılaştırılmıştır. Doğruluk, doğru tahmin edilen verilerin toplam veri kümesine oranlayarak başarı değerlendirmesi yaparken OKDO yöntemi pozitif olarak tahminlenen değerlerin gerçekte kaç adedinin pozitif olduğunu gösteren Kesinlik değerlerinin iki kere ortalaması alarak başarıyı değerlendirmektedir. Başarı değerlendirmeleri sonucunda, iyileştirilmiş yönteminin her iki kriter yönünden de daha başarılı olduğu gözlenmiştir.

Mei ve Gül (2020) veri üretmek için kullanılan Çekişmeli Üretici Ağ modelinin farklı bir yöntemi olan koşullu Wasserstein Çekişmeli Üretici Ağ ve görsel içerisindeki pikseller arasında bağlantı bularak resim içerisindeki nesneyi tespit eden bağlantı haritaları yöntemlerini birleştirerek yol çatlak tespiti için yeni bir metot önermişlerdir. Bu yöntemin yanında bir araca GoPro spor kamerası entegre ederek yeni yol çatlak verisi elde

etmişlerdir. Önerilen yöntem özellikle pikseller üzerinde çatlak tespiti yapabilmesi nedeniyle; Duyarlılık, Kesinlik ve F1-score açısından başarısı diğer mevcut yöntemlerle kıyaslandığında oldukça yüksek performansa sahip olduğu ve yenilikçi bir yöntem geliştirdikleri görülmektedir.

Naddaf-Sh, Naddaf-Sh, Kashani ve Zargarzadeh (2020) çalışmasında DSA tabanlı gerçek zamanlı olarak çatlak tespiti yapabilen bir yöntem geliştirmişlerdir. Geliştirilen yöntem mobil cihazlarda kullanılarak saniyede yaklaşık 180 resim tespit edebilmiştir. Önerilen yöntem F1-score yönünden değerlendirildiğinde ise ortalama %54 oranında başarılı çatlak tespiti yapabilmiştir.

Sizyakin, Voronin, Gapon ve Pižurica (2020) tarafından yapılan çalışmada doğru bir şekilde çatlak tespiti için biyomedikal görüntü segmentasyonu için geliştirilmiş Evrişimli Sinir Ağı tekniklerinden biri olan U-Net (Ronneberger, Fischer ve Brox, 2015) kullanmışlardır. U-Net ağ yapısı tarafından lokalize edilen çatlaklar ve morfolojik fitreler kullanılarak yeni bir binary haritası geliştirmişlerdir. Bu önerilen çözüm sayesinde geleneksel çatlak tespit yöntemlerine göre daha gerçekçi şekilde çatlaklarını konumu belirlenebilmektedir.

Han ve diğerleri (2021) yaptıkları çalışmada Derin Öğrenme algoritmalarından biri olan Evrişimli Sinir Ağlarını kullanarak otonom bir görüş tabanlı bir çatlak tespit yöntemi önermişlerdir. Bu yöntem ile herhangi bir ön süreç olmadan doğrudan resimlerdeki çatlakların özelliklerinin öğrenimi sağlanmaya çalışılmıştır. Çeşitli görsel verileri üzerinde denemeler yapılmış ve %96 oranında oldukça başarılı bir yöntem sunulmuştur.

Arya, Maeda, Ghosh, Toshniwal ve Sekimoto (2021) çalışmasında çeşitli ülkelerden elde edilen farklı yol hasarlarını bulunduran toplam 26.336 adet yol görseli içeren RDD2020 olarak adlandırdıkları yeni bir veri seti oluşturmuş ve DSA'nı kullanarak yeni bir yöntem sunmuşlardır. Görsel verileri toplamak amacıyla önerilen yöntemi kullanan bir akıllı telefon uygulaması geliştirilmiş ve telefonu bir kamera gibi kullanarak her saniyede bir resim yakalanmıştır. Elde edilen görsel veri seti içerisindeki çatlak tipleri farklı olarak

etiketlenmiş ve sınıflandırılmıştır. Sonuç olarak görsel içerisinde çatlağın konumu ve türü tespit edilebilmiştir.

Yukarıdaki çalışmalara bakıldığında araştırmacılar tarafından çok çeşitli çatlak tespit yöntemlerinin önerildiği gözlenmektedir. Ancak çoğunlukla sunulan yöntemlerde çatlak tespitinde sadece yüksek doğruluğa odaklanılmıştır. Dolayısıyla gerçek zamanlı çatlak tespitini amaçlayan çalışma sayısı azdır ve bu eksikliğe katkıda bulunulması gerekmektedir. Diğer taraftan önerilmiş gerçek zamanlı çatlak tespit yöntemlerine bakıldığında genel olarak yüksek teknik donanımlar gerekmektedir ve bu tür donanımların gerçek hayatta kullanılabilmesi için çok büyük bütçeler harcanmalıdır. Bu çalışmada, maliyeti olabildiğince azaltmak amacıyla gerçek zamanlı yüksek doğrulukta çatlak tespiti yaparken minimum donanım gerektiren bir model tasarlanmaya çalışılmıştır.

Önerilen yöntemin mevcut yöntemlerden ayrılan yönlerine bakıldığında; ilk kez çatlak tespiti çalışmalarında GYBE_DSA yöntemi uygulanmakta ve çatlak tespiti için önceden DSA kullanılarak elde edilmiş parametreler görüntü işleme teknikleriyle birlikte kullanılmaktadır. Ayrıca yapılan çalışmaların aksine bu çalışmada gerekli donanım gücünü minimum da tutarak piksel bazlı çatlak tespiti yerine alan tabanlı çatlak tespiti yapılmaktadır.

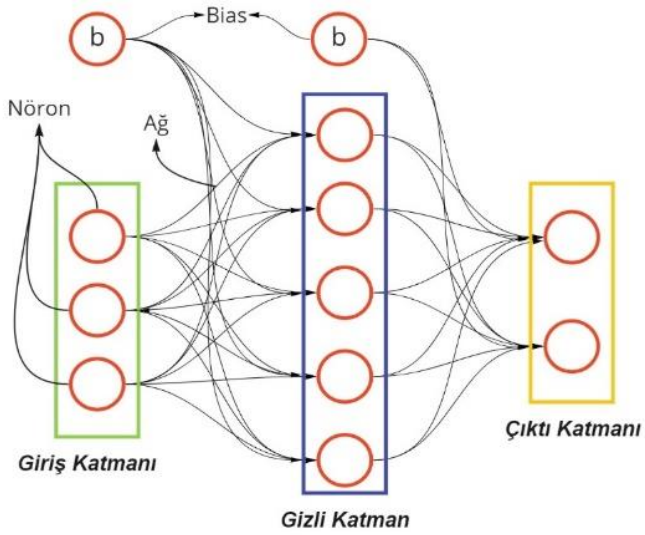
3. MATERYAL ve YÖNTEM

Bu bölümde çalışmada kullanılan materyaller ve yöntemlere detaylıca değinilmiştir. Ancak bu kısımlara geçmeden önce çalışma kapsamında kullanılan bazı kavram ve yöntemler hakkında genel bilgiler verilmiş ve ayrıca önerilen yöntemlerin kısa bir özeti sunulmuştur.

3.1. Yapay Sinir Ağları

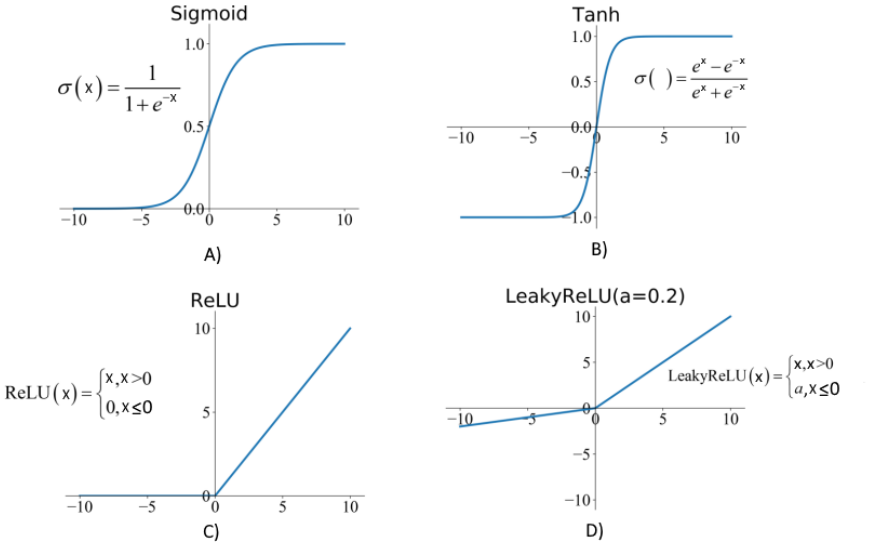
Araştırmacıların mevcut teknoloji ve yöntemleri yetersiz bulması ve sürekli farklı modeller keşfetme isteği yeni teknik ve alanların ortaya çıkmasını sağlamıştır. Günümüzde hala önemli bir yere sahip olan bu alanlardan biri de insan beyninin öğrenme yolunu taklit eden Yapay Sinir Ağları (YSA) tekniğidir. İlk YSA modeli 1943 yıllında bir sinir hekimi Warren McCulloch ve bir matematikçi Walter Pitts tarafından yayınlanan bir makalede (McCulloch ve Pitts, 1943) ortaya çıkmıştır. Ancak YSA kullanım açısından 1990'lara kadar aktif bir ivme yakalayamamıştır. Bu tarihten itibaren hızla gelişmeye başlamış ve bugün itibariyle önemli bir yere sahip olmuştur.

YSA temel olarak giriş katmanı, gizli katman veya katmanlar ve çıktı katmanından oluşmaktadır. Her bir katmanda Şekil 3.1'de gösterildiği üzere bilgi saklayan nöronlar ve nöronları birbirine bağlayan ağlar bulunmaktadır. Ayrıca nöronlar arasında iletişimi sağlayan her bir ağ için bir ağırlık değeri ve her bir katman için yerel optimuma yakalanma riskini azaltan sabit bir değer içeren yanlılık değeri mevcuttur.



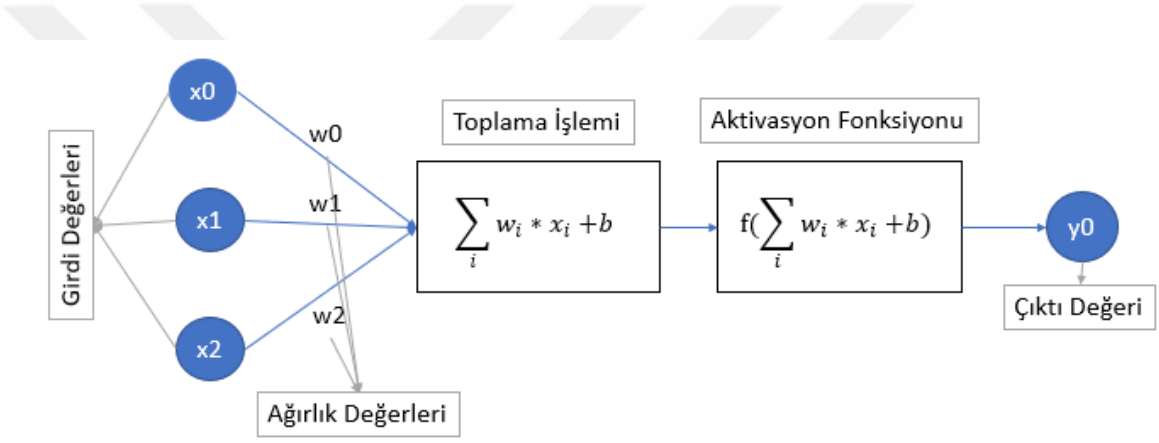
Şekil 3.1. Temsili yapay sinir ağı

Katmanlar içerisinde bulunan nöronlar diğer katmandaki nöronlar ile etkileşime girip bilgi paylaşmaktadır. Ancak bilgi aktarımı yapılırken aktivasyon fonksiyonu olarak adlandırılan doğrusal olmayan bir matematik fonksiyon kullanılmaktadır. Böylece aktarılan her bilgi aktivasyon fonksiyonuna girip güncellendikten sonra aktarılacak nörona işlenmektedir ve Şekil 3.2'de genellikle kullanılan aktivasyon fonksiyonları gösterilmektedir. Bu çalışma kapsamında her iki katmanda da Sigmoid fonksiyonu kullanılmıştır.



Şekil 3.2. Genellikle kullanılan aktivasyon fonksiyonları (Feng ve diğerleri, 2019)
A) Sigmoid, B) Tanh, C) ReLU, D) LReLU

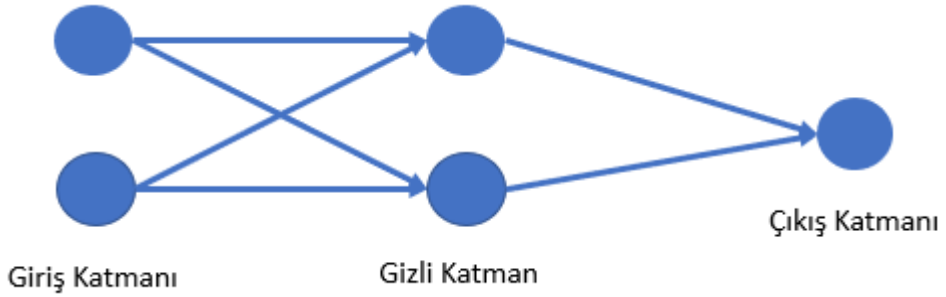
Tüm aktarım sürecinden sonra elde edilen çıktı değerleri, belirlenen sınıflar ya da sayılarla karşılaştırılmakta ve hata miktarı elde edilmektedir. Ardından bu hata değerleri dikkate alınarak ağların sahip olduğu ağırlık değerleri güncellenmektedir ve bu işlem hata değeri belirlenen sınır değerine ulaşana kadar sürekli tekrar etmektedir. Şekil 3.3'te basit bir ağ yapısının elemanları ve öğrenim süreci gösterilmektedir. Sistem içinde tanımlanan x0, x1 ve x2 girdi değerleri çıktı hücresiyle bağlı olduğu ağlara ait w0, w1 ve w2 ağırlık değerleriyle çarpılmakta ve tüm elde edilen değerler toplanmaktadır. Ardından hesaplanan değere sabit bir sayı içeren b yanlılık terimi eklenmektedir. Son olarak bu değerler $f(\sum_i w_i * x_i + b)$ aktivasyon fonksiyonu tarafından işleme tabi tutulmakta ve çıktı değeri elde edilmektedir.



Şekil 3.3. Basit bir ağ yapısının öğrenim süreci

3.1.1. İleri Beslemeli Yapay Sinir Ağı Modeli

İleri beslemeli YSA'da Şekil 3.4'te gösterildiği üzere giriş katmanından çıktı katmanına doğru bir akış söz konusudur ve her bir katmanın sadece kendinden sonraki katman ile ağ bağlantısı bulunmaktadır. Dolayısıyla giriş katmanında bulunan bilgiler gizli katmana iletilmekte ve aktivasyon fonksiyonuna maruz kalmaktadır. Ardından gizli katmanda elde edilen veriler çıktı katmanına aktarılmakta tekrar aktivasyon fonksiyonu ile işleme tabi tutulmakta ve çıktı değeri elde edilmektedir. Son olarak elde edilen çıktı değeri ile gerçek değer karşılaştırılmakta ve Denklem 3.1'de hesaplanan hata değerine bağlı olarak giriş katmanındaki ağırlık değerleri Denklem 3.2 kullanılarak güncellenmektedir.



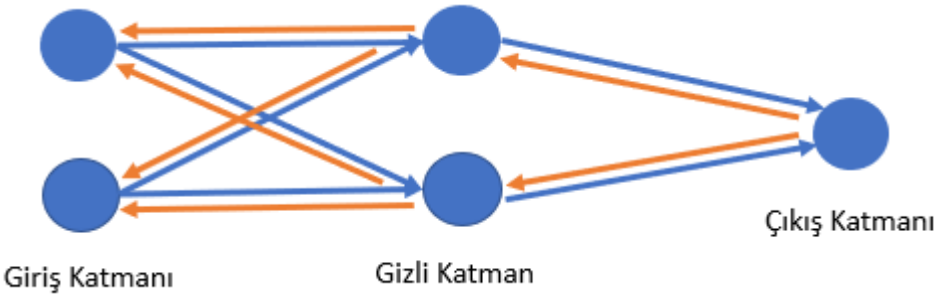
Şekil 3.4. İleri beslemeli yapay sinir ağı

$$Hata^2 = \left(y - f(\sum_i w_i * x_i + b)\right)^2 \tag{3.1}$$

$$W_y = W_e + a * Hata * f'\left(\sum_i w_i * x_i\right) * x_i \tag{3.2}$$

3.1.2. Geri Beslemeli Yapay Sinir Ağı Modeli

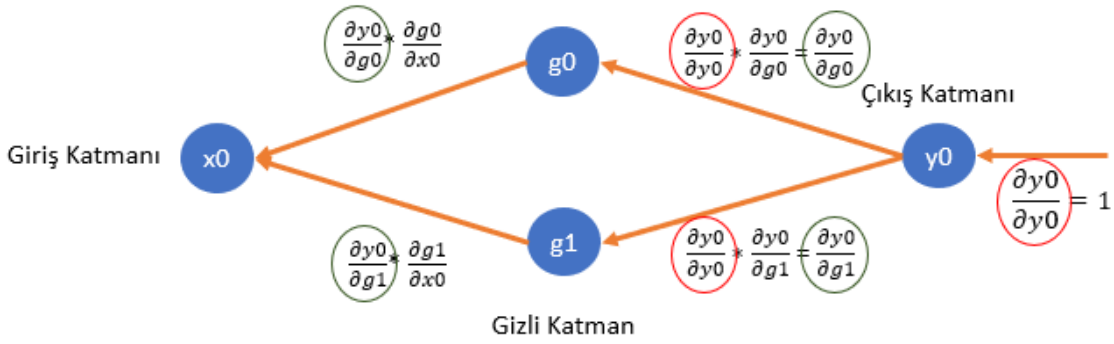
Geri beslemeli veya geri yayılımlı YSA'da ileri beslemeli YSA'dan farklı olarak hata değeri elde edildikten sonra Şekil 3.5'te gösterildiği üzere ağırlık parametreleri giriş katmanına kadar geriye yönelimli olarak güncellenmektedir. Bu sayede tahmin edilen çıktı değerinin gerçek çıktı değerine daha da yakınlaşması sağlanmaktadır.



Şekil 3.5. Geri beslemeli yapay sinir ağı

Geri yayılımlı güncellemeden önce ileri doğru beslemede aktivasyon fonksiyonu kullanılmaktadır. Geri beslemede ileri doğru besleme sonucu elde edilen verileri kullanabilme amacıyla ağırlıkların geriye doğru güncellemesinde aktivasyon

fonksiyonun kısmi türevi kullanılmaktadır. Böylece hızlı bir şekilde geriye yayımlı olarak ağırlık güncellemesi yapılabilmektedir. Güncelleme temel olarak Şekil 3.6'da gösterildiği gibi olmaktadır. Çıkış katmanındaki hücrenin gizli katman dışındaki hücreler dışında başka bir hücreyle bağlantısı olmadığı için çıktı değerinin kendine göre kısmi türevi alınmaktadır. Gizli katman ile çıktı katmanı arasındaki ise çıkış katmanında elde edilen ve şekil içerisinde kırmızı daire ile gösterilen kısmi türev değeri çıkış katmanındaki hücrenin değeri gizli katmandaki hücrelerin değerlerine göre kısmi türeviyle çarpılmaktadır. Şekil üzerinde yeşil daire ile belirtilen ağırlık değerleri bulunduktan sonra bu değerler gizli katmandaki hücrelerin değerlerinin giriş katmanındaki hücrenin değeriyle çarpılmakta ve giriş katmanı ile gizli katman arasındaki ağırlık parametrelerin değerleri elde edilmektedir. Bu aşamadan sonra bu ağırlık değerleri kullanılarak ileri besleme yapılmakta ve bu öğrenim istenilen hata değerine ulaşana kadar devam etmektedir.



Şekil 3.6. Geri yayımlı bir ağda ağırlık parametrelerinin güncellenmesi

3.2. Derin Sinir Ağları

YSA yukarıda bahsedildiği üzere oldukça başarılı bir yöntem olsa da özellikle her sisteme uygulanamaması nedeniyle tek başına yetersiz kalmaktadır. Bu sorunu çözmek için DSA geliştirilmiştir. DSA temel olarak çok fazla gizli katmanında oluşan veya çok fazla nöron bulunduran yapay sinir ağlarıdır. Çok katmanlı yapısı sayesinde çok karmaşık problemleri çözerken yüksek verimliliğe sahiptir ve bu durum DSA tekniğini oldukça popüler bir yöntem haline getirmiştir. Günümüzde araştırmacılar tarafından DSA

kapsamında Çok Katmanlı Algılayıcılar, Evrişimli Sinir Ağları, Yinelemeli Sinir Ağları ve Çekişmeli Sinir Ağları olmak üzere çeşitli teknikler bulunmaktadır.

İlk yöntem girdi, gizli katman veya katmanlar ve çıktı katmanı içeren Çok Katmanlı Algılayıcılar veya diğer bir adıyla Çok Katmanlı İleri Beslemeli Ağlar (Hornik, Stinchcombe ve White, 1989) metodudur. Ağ parametrelerini hem ileriye hem geriye doğru güncelleyen Çok Katmanlı Algılayıcılar, sınıflandırma ve genelleme yapma durumlarında kullanılmaktadır. İkinci olarak çeşitli yapılarda gizli katmanlara sahip olan ve katmanlar içerisinde eğitim yapılan görsele matris operasyonları uygulayan Evrişimli Sinir Ağları bulunmaktadır. Evrişimli Sinir Ağları (LeCun, Bottou, Bengio ve Haffner, 1998) genel olarak görsel bilgilerin analiz edilmesi gereken alanlarda uygulanmaktadır.

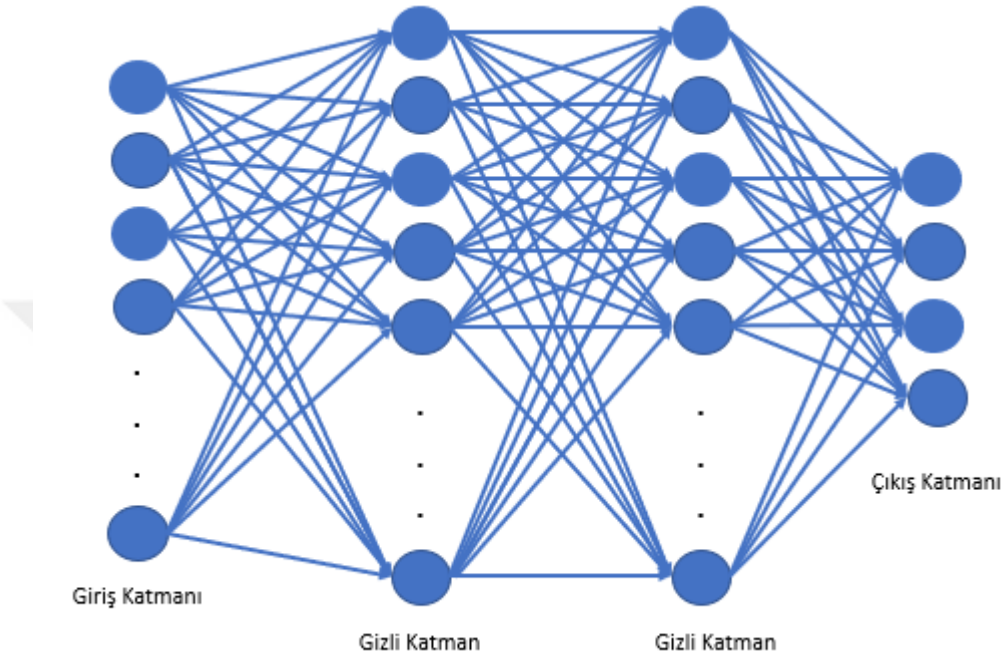
Yinelemeli Sinir Ağları (Karjala, Himmelblau ve Miikkulainen, 1992) ise üçüncü yöntemdir. Bu yöntem içerisinde her bir parametrenin değeri sistem içerisinde kaydedilmekte ve hata değerine göre parametre güncellenmesinde bu değerlerde dikkate alınmaktadır. Yinelemeli Sinir Ağları çoğunlukla yazı ve konuşma tanıma amacıyla kullanılmaktadır. Son ve en güncel yöntem ise Çekişmeli Üretici Ağlar tekniğidir. Yöntem içerisinde iki farklı yapay sinir ağı bulunmaktadır. Bu ağlardan biri sahte veriler üretirken diğeri bu verileri sahte ve gerçek veri olarak ayırt etmeye çalışılmakta ve bu şekilde öğrenim devam etmektedir. Çekişmeli Üretici Ağlar (Goodfellow ve diğerleri, 2014) genel olarak yeni görsel veriler üretilmek istenen çalışmalarda kullanılmaktadır.

Bu çalışmada da genelleme ve sınıflandırma işlemi için en ideal yöntem olan Çok Katmanlı Algılayıcılar yöntemi kullanılarak yüksek doğrulukta çatlak tespiti yapılması amaçlanmıştır.

3.2.1. Çok Katmanlı Algılayıcılar

Çok Katmanlı Algılayıcılar bir dizi girdi verisini işleyerek bir dizi çıktı verisi elde eden YSA çeşididir ve YSA modelinin temellerini oluşturan tekniktir. Ağ yapısı içerisinde giriş katmanı, gizli katman ve çıktı katmanı olmak üzere 3 katman bulunmaktadır. Ancak bu ağ yapısı çok fazla girdi sayısına sahip sistemlerde iyi bir öğrenim için yeterli katman

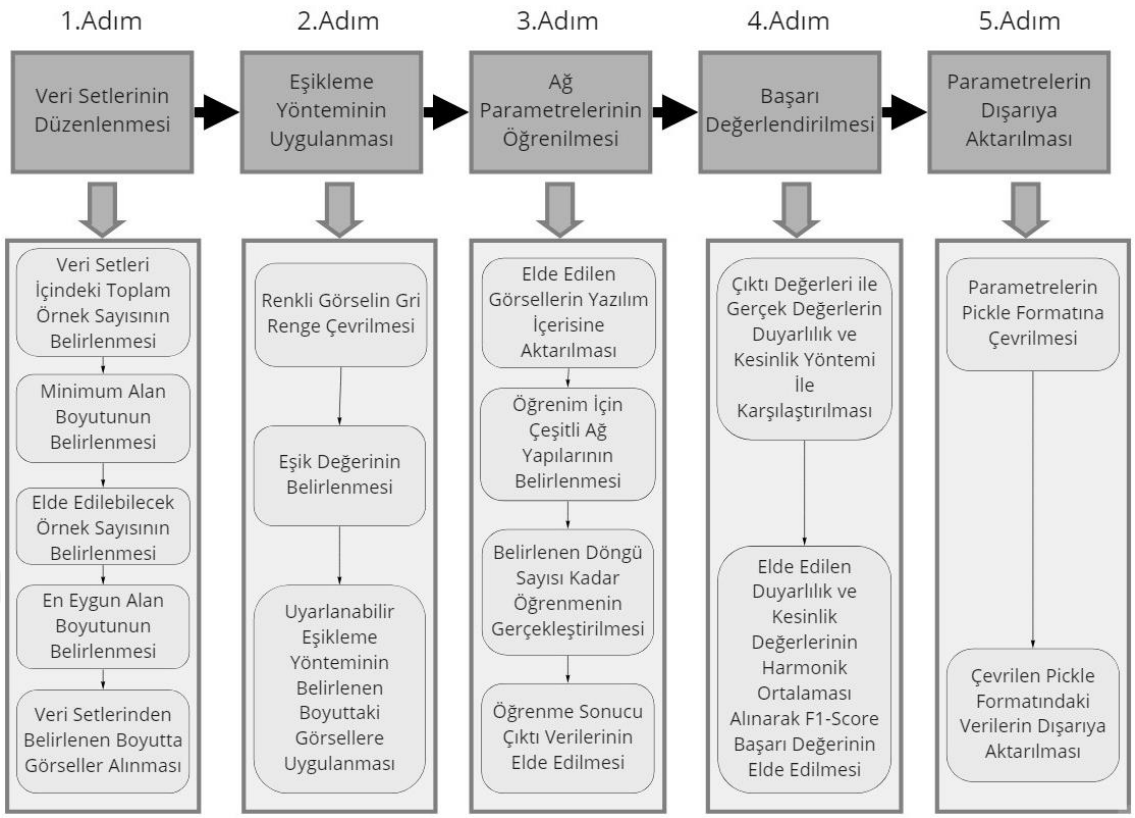
ve nöron sayısına sahip değildir. Bu nedenle DSA fikri bu ağ yapısı içinde kullanılmıştır ve bulundurduğu elemanların sayısının artırılması önerilmiştir. Dolayısıyla DSA kapsamında kullanılan ve Şekil 3.7'de örnek gösterildiği gibi daha karmaşık bir ağ yapısına sahip Çok Katmanlı Algılayıcılar yöntemi geliştirilmiştir.



Şekil 3.7. Çok Katmanlı Algılayıcılar yönteminin temsili bir ağ yapısı

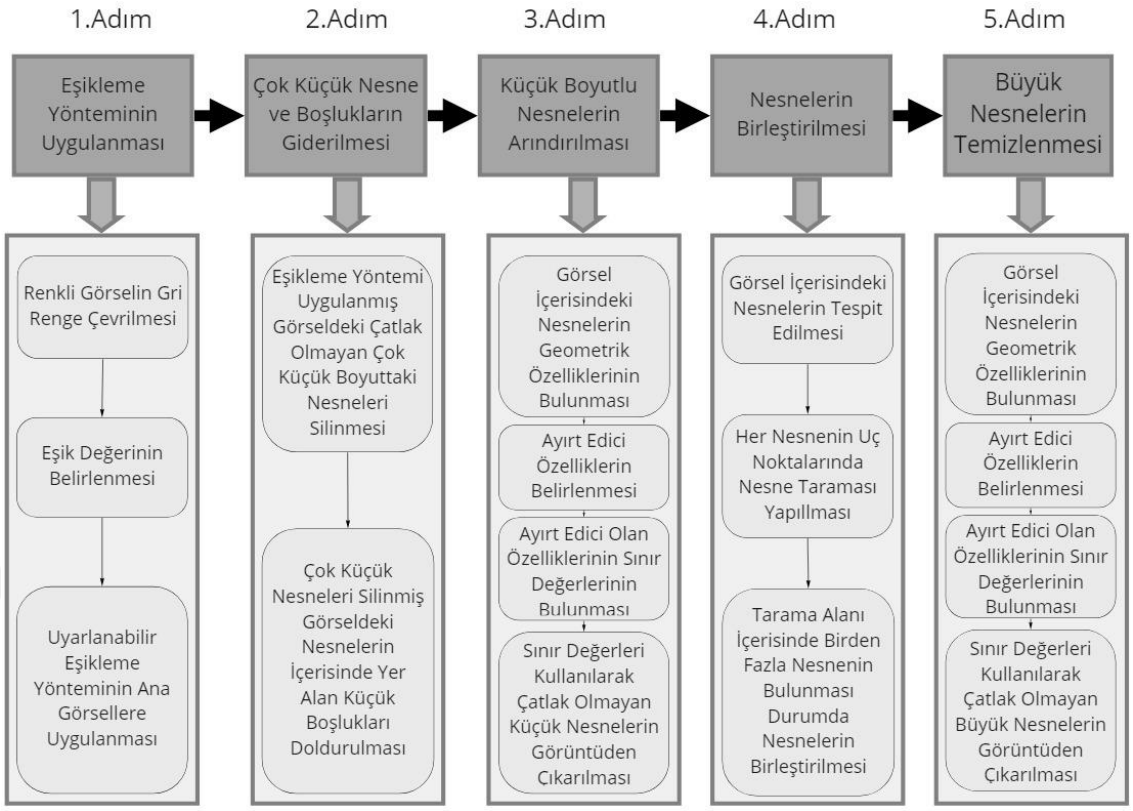
3.3. Önerilen DSA Tabanlı Çatlak Tespit Yöntemi

Daha önce Şekil 1.1'de verildiği üzere önerilen yöntem her bir sürecin 5 aşamalı olduğu 2 süreçten oluşmaktadır. İlk sürece ait adımlar daha detaylı olarak Şekil 3.8'de sunulmuştur. Bu süreç içerisindeki 1.Adım'da çeşitli veri setlerinden yararlanarak yeni bir veri seti elde edilirken 2.Adım'da bu veri setine eşikleme yöntemi uygulayarak gereksiz işlem karmaşıklığı azaltılmaktadır. Ardından 3.Adım'da GYBE_DSA yaklaşımıyla geliştirilmiş ÇKA kullanılarak her bir ağ yapısının ağırlık ve yanlılık parametreleri elde edilmektedir. Sonraki 4.Adım'da gerçek değerler ile tahmin değerleri karşılaştırarak başarı değerlendirmesi yapan Kesinlik, Duyarlılık ve F1-Score yöntemleri kullanılmaktadır. Son olarak 5.Adım'da ise en yüksek başarıya sahip olan parametreler ikinci süreçte kullanılmak üzere dışarıya aktarılmaktadır.



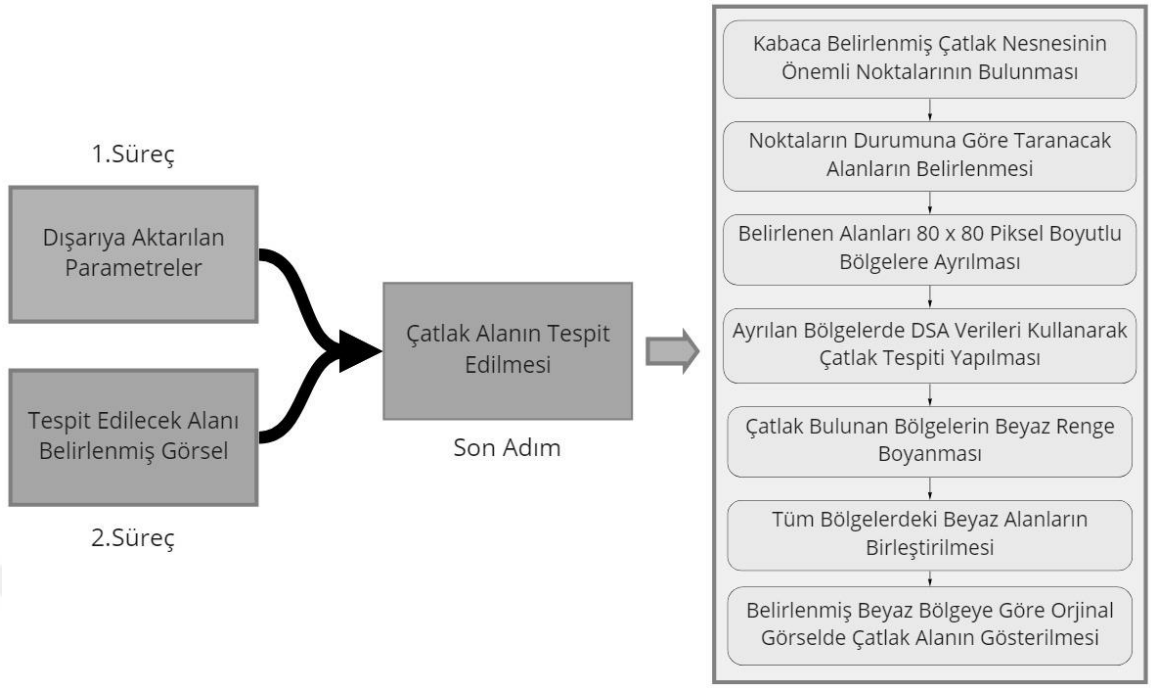
Şekil 3.8. Önerilen yöntemdeki birinci sürecinin akış şeması

İkinci süreçte Şekil 3.9'da gösterilen akış şeması izlenmektedir. Şemada belirtildiği üzere 1.Adım'da eşikleme yöntemi uygulayarak siyah beyaz bir görsel elde edilmekte ve ardından diğer 4 adımda belirtilen yöntemler kullanılarak çatlak olmayan nesnelerden görsel arındırılmakta ve çatlağın konumu kabaca belirlenebilmektedir.



Şekil 3.9. Önerilen yöntemdeki ikinci sürecinin akış şeması

Son aşamada ise Şekil 3.10'da gösterilen akış şeması izlenmektedir. 1.Süreçte GYBE_DSA yöntemi kullanılarak elde edilen parametreler 2.Süreçte belirlenen alanda çatlak tespiti yapılmak için kullanılmaktadır. Sonuç olarak önerilen yöntem sayesinde bir görsel üzerinde çatlak bulunduran bölgeler belirlenebilmektedir.



Şekil 3.10. Önerilen yöntemdeki son aşamanın akış şeması

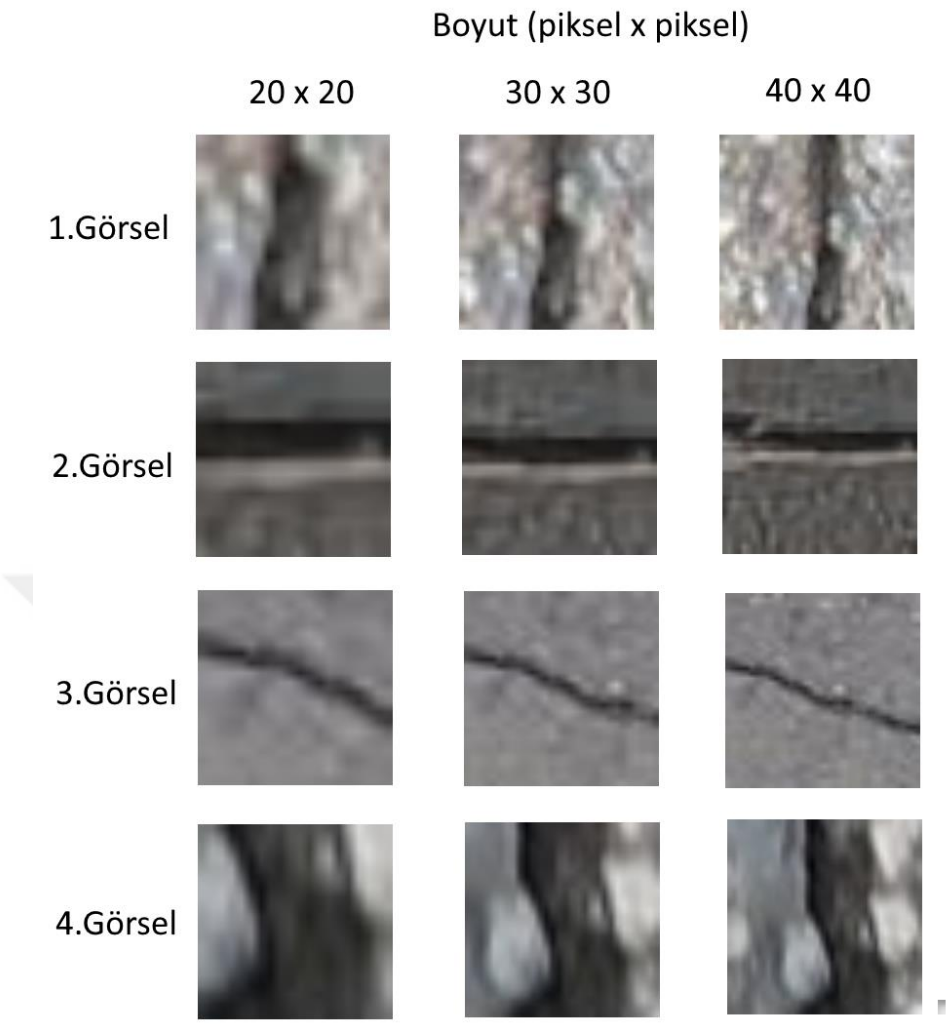
3.3.1. Veri Seti

Bu çalışmada, çatlak tespiti ile ilgili makalelerin çoğunda kullanılmış ve yol çatlağıyla alakalı çeşitli sayıda ve çözünürlükte görseller içeren Çizelge 3.1'de belirtilen EdmCrack600 (Mei, Gül, ve Azim, 2020; Mei, Gül, ve Shirzad-Ghaleroudkhani, 2020; Mei ve Gül, 2020), AsphaltCrack (Jayanth Balaji, Thiru Balaji, Dinesh, Binoy ve Harish Ram, 2019), CFD ve CrackSegmentation (Amhaz, Chambon, Idier ve Baltazart, 2016; Eisenbach ve diğerleri., 2017; Zhang, Yang, Daniel Zhang ve Zhu, 2016; Zou, Cao, Li, Mao ve Wang, 2012) veri setleri kullanılmıştır.

Çizelge 3.1. Çalışmada kullanılan veri setlerinin örnek sayısı ve görsel boyutu

Veri Seti	Örnek Sayısı	Görsel Boyutu (px*px)
EdmCrack600	600	1920x1080
AsphaltCrack	200	448x448
CFD	155	480x320
CrackSegmentation	428	448x448

Veri setleri içerisinden hangi boyutta resim parçaları alınacağı önemli bir konudur ve boyutun değerini sınırlandıran iki temel etken vardır. İlk olarak alınan bölgede çatlak tespiti yapılacağı için görsel içerisinde çatlağın net bir şekilde gözükmesi gerekmekte yani çatlağın en az bir boyutunun bölge içerisinde uygun bir konumda olmalıdır. Bu nedenle Şekil 3.11'de gösterildiği üzere veri setlerinden 4 farklı görsel içerisinden 20 x 20 piksel, 30 x 30 piksel, 40 x 40 piksel boyutlarında çatlak görselleri alınmıştır. Elde edilen görsellere özelikle ise 4.Görsel'e dikkat edilirse 20 x 20 piksel ve 30 x 30 piksel boyutlarındaki görseller içerisindeki çatlağın alanı görselin tüm alanının yarısından fazlasını oluşturmaktadır ve bu durumda ana görseldeki çatlağı temsil edecek görsel veriler elde edilemez. Dolayısıyla veri setlerinden alınacak görsellerin en az 40 x 40 piksel boyutunda olmasının daha uygun olduğuna karar verilmiştir.



Şekil 3.11. Veri setlerinde yer alan görsellerden 3 farklı boyutta görsel çıkarılması

Diğer taraftan bir GYBE_DSA yönteminde öğrenilmeye başlanılması için girdi değerlerine yani öğrenilecek verinin belirleyici özelliklerine ihtiyacı vardır. Bu nedenle bir görsel üzerinde öğrenim yapılmak istendiğinde görselin özelliklerinin bilinmesi gerekmektedir ve bir görselin özellikleri her bir pikselinde bulundurduğu değeri ifade etmektedir. Dolayısıyla bir görselin boyutu ne kadar büyük olursa öğrenme işleminde o kadar fazla özellik girdisi kullanılmaktadır ama örnek sayısının en az özellik sayısı kadar olmasının öğrenmeye olumlu açıdan katkıda bulunacağı da düşünülmektedir. Bu sebepten ötürü seçilecek boyuta göre gereken minimum örnek sayısı Çizelge 3.2'de gösterildiği üzere bulunmuştur. Ancak elde edilebilecek görsel sayısı tam olarak bilinmediği için veri setlerindeki her bir görselden yaklaşık olarak 8 adet belirli bir boyutta çatlak görseli elde edilebileceği varsayımda bulunulmuştur. Tüm veri setleri içinde toplam 1.000 tane görsel

verisi bulunması nedeniyle maksimum 8.000 tane belirli bir boyutta çatlak içeren görseller elde edileceği belirlenmiştir.

Çizelge 3.2. Belirlenen boyutlara bağlı olarak gereken minimum örnek sayısı

Belirlenen Boyut	Girdi Sayısı	Minimum Örnek Sayısı
40 x 40	40*40=1.600	1.600
50 x 50	50*50=2.500	2.500
60 x 60	60*60=3.600	3.600
80 x 80	80*80=6.400	6.400
100 x 100	100*100=10.000	10.000
120 x 120	120*120=14.400	14.400
150 x 150	150*150=22.500	22.500

Dolayısıyla 8.000 örnek verisinden daha az örnek verisine ihtiyacı olan maksimum boyuta sahip boyut değeri belirlenmelidir. Bu kapsamda, en az 6.400 örnek verisi gerektiği düşünülen 80 x 80 boyuta sahip görseller alınmasının en uygun seçim olduğuna karar verilmiştir. Ek olarak belirlenen boyutta çatlak içeren görseller elde etmenin çatlak içermeyen görseller elde etmekten daha belirleyici ve zorlu bir süreç olduğu için belirlenen boyutta çatlak içermeyen görsellerin durumu dikkate alınmamıştır.

Sonuç olarak, görsel veri setleri içerisinde 80 x 80 piksel boyutunda görsel parçalar alınarak kaydedilmiştir. Böylelikle Şekil 3.12'de örnek olarak gösterilen toplam 15.000 tane 80 x 80 piksel boyutunda çatlak içeren ve içermeyen görseller elde edilmiştir. Ek olarak çatlak içeren ve içermeyen görsellerin sayısı özellikle eşit olacak şekilde her biri için 7.500 tane görsel bulunmaktadır.



Şekil 3.12. Rastgele olarak alınmış çatlak içeren ve içermeyen 80x80 piksel boyutunda görseller

3.3.2. Geri Yayılımlı Birlikte Evrim ile Öğrenme Süreci

Çalışma için aktarılan veriler renkli görseller olup sırasıyla; eğitim ve test veri setleri ayrı ayrı olarak kod içerisine aktarılmıştır. Ancak bahsedildiği üzere öğrenme aşamasında her bir girdi değeri ağırlık değeri ile çarpılmakta ve renkli veya gri renkli görsellerde her bir piksel çok farklı değerler alabilmektedir. Dolayısıyla girdi değeri ne kadar büyük ve farklı değerler alırsa o kadar fazla matematiksel hesaplamaya neden olmaktadır ve bu durum öğrenmeyi negatif yönde etkilemektedir. Bu yüzden öğrenme yapılacak görseldeki her bir pikselin olabildiğince işlemi basitleştiren değerler alması işlemleri kolaylaştıracaktır. Bu açıdan en ideal resim tipi siyah-beyaz resimler olacaktır. Zira siyah-beyaz resimler her bir piksel için sadece 0 ve 255 değeri alabilmektedir. Yine de 255 değeri büyük bir değerdir ve ağırlık değerleri ile çarpılması sırasında çok fazla değere sahip virgüllü veya tamsayı değerler elde edilir. Bu nedenle daha basitleştirme yapılmalıdır ve bu amaç için en iyi yolda siyah beyaz bir resmin her pikseli 255 bölerek sadece 0 ve 1 değerlerini içeren görseller oluşturulabilir. Böylece öğrenme işlemi sırasında bir değerle çapıldığında o sayıyı yok eden 0 ve etkisiz olup herhangi bir değişim yapmayan 1 değerlerinin kullanılması harcanan zamanı olabildiğince azaltmaktadır.

Renkli bir resme eşikleme yöntemleri uygulayarak siyah-beyaz bir resim kolaylıkla elde edilebilir. Eşikleme yöntemleri temel olarak görsel içerisinde yer alan nesnelerin arka plandan ayrılmasını sağlayan yöntemlerdir. Açık kaynaklı görüntü işleme kütüphanesi olan OpenCV kütüphanesinde İkili Eşikleme, Otsu Eşikleme ve Uyarlanabilir Eşikleme gibi çeşitli eşikleme yöntemleri mevcuttur ve çoğu yöntem belirli bir amaç ve ortam koşullu için kullanılmaktadır. Çalışmada kullanılacak olan veri setlerindeki görseller genel olarak farklı alanlarda farklı ışık koşullarında elde edilmiş görsellerdir. Dolayısıyla bu ortam koşullarına göre en iyi eşiklemeyi yapabilen Uyarlanabilir Eşikleme Yönteminin kullanılması daha uygun olmaktadır. Ancak eşikleme yöntemlerinin sadece gri renkteki resimlere uygulanabileceği için eşikleme uygulanacak görseller önce gri renge çevrilmelidir. Bu amaçla PHYTON yazılımı içerisinde renkli bir görseli gri renge çevirmek için Şekil 3.13'te verilen iki yazılım kodu kullanılabilir.

```
1.Yöntem
-----------------------------------------------
import cv2

# path='Görselin bilgisayar içerisindeki konumu'

color_image= cv2.imread(path)

gray_image = cv2.cvtColor(image, cv2.COLOR_BGR2GRAY)

#####################################################
2.Yöntem
-----------------------------------------------
import cv2

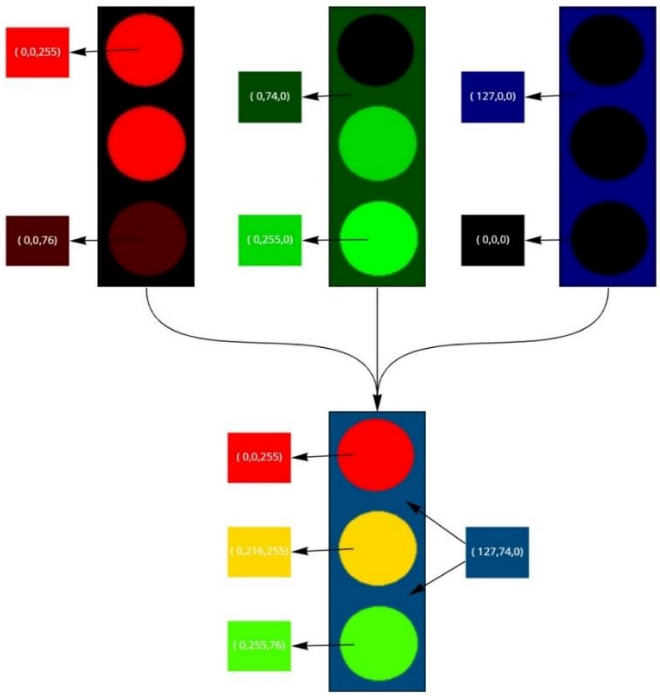
# path='Görselin bilgisayar içerisindeki konumu'

gray_image= cv2.imread(path,0)
```

Şekil 3.13. Renkli görseli gri renge çevirmek için yazılım kodları

Renkli bir resmi gri renge çevirme işleminden sonra gri renkli resmin siyah-beyaz bir resme çevrilmesi gerekmektedir ancak bu sürece geçmeden önce konunun daha iyi anlaşılması için renkli ve gri renkli görseller hakkında kısa bir açıklama yapılmalıdır.

Renkli bir resim Şekil 3.14'te gösterildiği gibi kırmızı, yeşil ve mavi olmak üzere 3 farklı renk uzayının üst üste gelmesiyle oluşmaktadır. Her bir piksel her bir uzay için 0 ile 255 arasında değer almaktadır. Dolayısıyla bir pikselde bu üç renk kombinasyonunu ifade eden 3 farklı değer yer almaktadır. En baştaki değer mavi renk uzayını, ortadaki değer yeşil renk uzayını ve en sondaki değer kırmızı renk uzayını temsil etmektedir ve her bir pikselin değeri (M, Y, K) ile temsil edilmektedir. Burada M mavi renk uzayının renk değerini, Y yeşil renk uzayının renk değeri ve K kırmızı renk uzayının renk değerini göstermektedir. Bir piksel içerisindeki değerlerde 0 değeri siyah rengi gösterirken 255 değeri renk uzaylarının en son ulaşabileceği renk tonunu göstermektedir. Arada kalan değerler ise her bir uzayın bulundurduğu rengin tonlarını göstermektedir. Üç renk uzayı birleştirildiğinde ise her bir uzayın değer yukarıda bahsedilen konumda piksel içerisinde yer alır ve renklerin kombinasyonuna göre rengini belirlenir. Örneğin renkli resmin ortasında bulunan sarı renk mavi ve renk uzaylarının aldığı değerler sonucu oluşmaktadır.



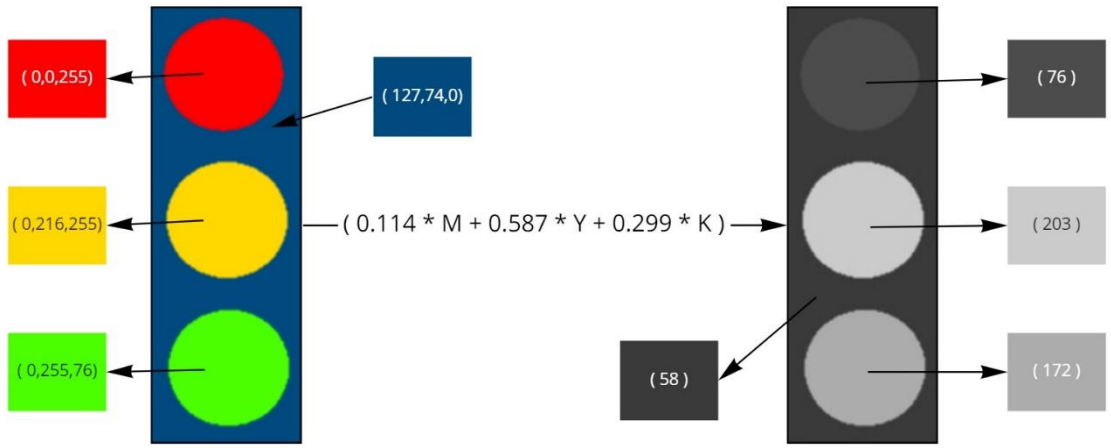
Şekil 3.14. Renkli bir görselin üç farklı renk uzayında görüntüsü ve piksel değerleri

Ancak eşikleme yöntemi tek bir renk uzayına sahip görsellerde kullanılabilmektedir ve dolayısıyla bahsedildiği üzere renkli resimler 3 farklı renk uzayına sahip olduğu için eşikleme yöntemi renkli resimlere uygulanamamaktadır. Bu nedenle öncelikle renkli

resimlerin gri renge yani tek bir uzaya indirgenmesi gerekmektedir. Bunun için Denklem 3.3'te verilen basit yöntem uygulanmaktadır. Yöntem içerisinde kısaca her piksel içerisinde yer alan her bir renk uzayına ait değer belirli bir sabit değerler çarpılmakta ve bulunan değerleri toplanıp tek bir değer elde edilmektedir. Elde edilen değeri ise işlem yapılan pikselin yeni değeri olarak belirlemektedir.

$$D = 0.114 * M + 0.587 * Y + 0.299 * K \quad (3.3)$$

Sonuç olarak her bir pikseli sadece bir uzaya sahip ve 0 ile 255 arasında toplam 256 farklı değer alabilen ve dolayısıyla 256 farklı siyah-beyaz tona sahip Şekil 3.15'te örnek olarak gösterilen gri renkte bir resim elde edilir.

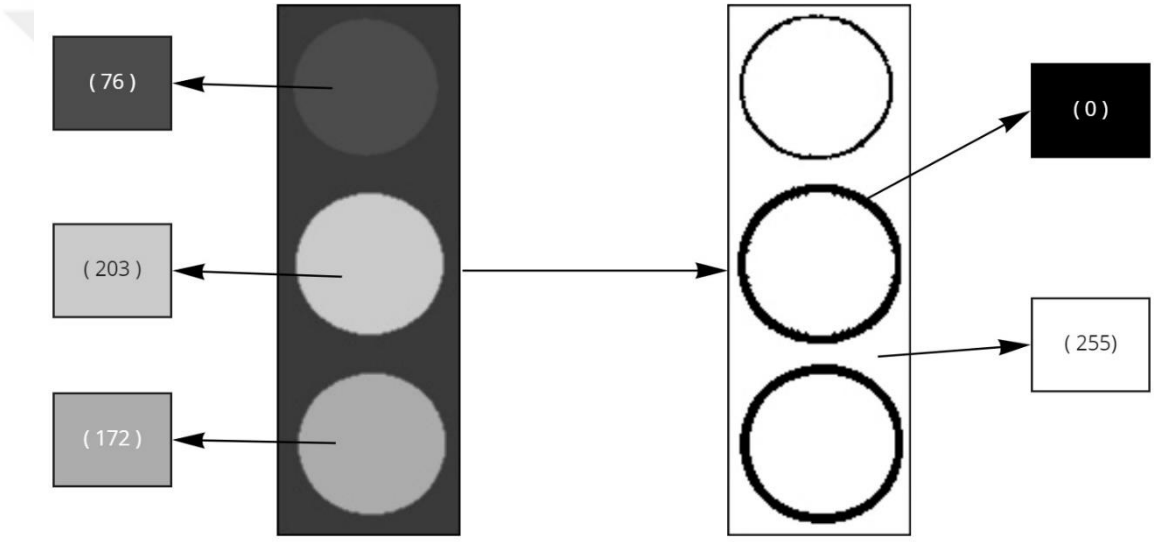


Şekil 3.15. Renkli bir görselin gri tona dönüştürülmesi ve piksel değerleri

Uyarlanabilir Eşikleme Yöntemi kullanılırken öncelikle 0 ile 255 değerleri arasında bir eşik değeri belirlenmelidir. Eşik değeri bir veriyi iki farklı sınıfa ayıran bir sınırlayıcıdır. Bu nedenle eşik değeri belirlenirken iki sınıfı en iyi şekilde ayıran değer seçilmelidir. Bu çalışmada da çatlak tespiti için kullanılacak CFD veri seti içerisindeki görsellere çeşitli eşik değerleri belirlenerek eşikleme yöntemi uygulanmış ve görseller içerisinde çatlağın en iyi şekilde diğer objelerden ayıran 135 değeri eşik değeri olarak seçilmiştir.

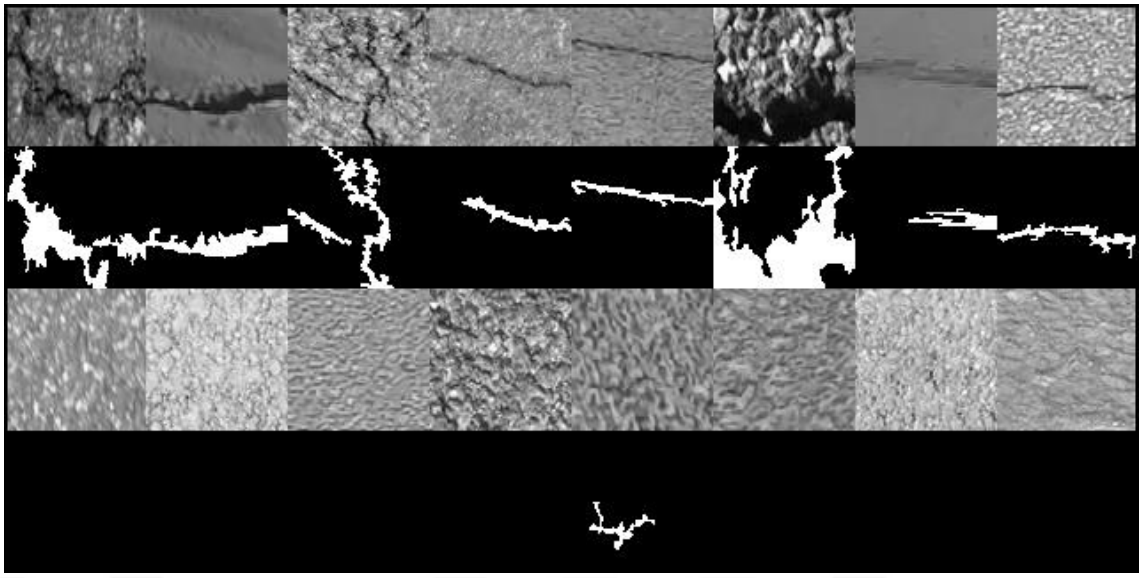
Eşik değeri belirlendikten sonra seçilen eşikleme yöntemi her bir pikselin çevresindeki piksellerin değerlerinin Gauss yöntemiyle ağırlıklı toplamların ortalamasını hesaplar ve

o pikselin hesaplanan değerini kaydeder. Böylece piksellerinin yanlarında bulunan piksellerin değerlerine bakılarak ortamın aydınlığı hakkında bilgi edinilmekte ve ortamın genel durumuna göre bir değer hesaplanmaktadır. Ardından her bir piksel için bulunan değerden belirlenen eşik değeri çıkarılır. Eğer hesaplanan değer eşik değerinden küçük olursa bu piksel arka plan olarak algılanır ve piksel 0 değerini yani siyah rengi alır. Ancak değer eşik değerine eşit ya da büyükse bu piksel nesne olarak algılanır ve piksel 255 değerini yani beyaz rengi alır. Sonuç olarak Uyarlanabilir Eşikleme Yöntemi kullanılarak Şekil 3.16'da gösterildiği üzere gri renkli bir resimden sadece 0 ve 255 değerleri alan siyah-beyaz bir görsel elde edilebilmektedir.



Şekil 3.16. Gri tonda bir resme Uyarlanabilir Eşikleme Yönteminin uygulanması

Çalışma kapsamında içeriye aktarılan resimlere Uyarlanabilir Eşikleme Yöntemi uygulayarak Şekil 3.17'de gösterilen resimler elde edilmiştir. Ardından eşikleme sonucu elde edilen verilere GYBE_DSA yöntemi ile öğrenme işlemi uygulanmış ve çeşitli sinir ağ yapıları kullanılarak farklı ağırlık ve yanlılık değerleri elde edilmiştir.

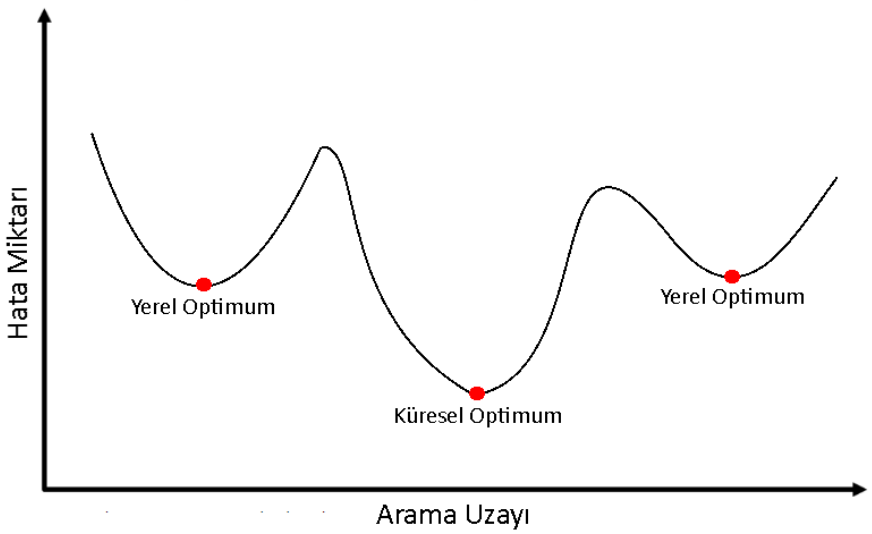


Şekil 3.17. Rastgele olarak seçilmiş 80x80'lik çatlak içeren ve içermeyen görsellere eşikleme yönteminin uygulanması sonucu elde edilen görsellere örnekler

3.3.3. Geri Yayılımlı Birlikte Evrim Yöntemi

Diğer yandan, GYBE_DSA yöntemin neden kullanıldığını açıklamak yöntemin anlaşılması açısından daha etkili olacaktır. GYBE_DSA yönteminin kullanılmasının en önemli nedeni Geri Yayımlı (GY) (Rumelhart, Hinton ve Williams, 1986) öğrenmede sıkça karşılaşılan yerel optimuma takılma sorunundan kurtulmak ve daha etkili bir öğrenim gerçekleştirmektir. Genel olarak GY yaklaşımıyla öğrenme sürecinde en sık karşılaşılan problem yerel optimuma takılmadır. Yerel optimum öğrenme sırasında belirli bir arama uzayı içerisindeki en az hata değerini ifade eder ancak öğrenme için en iyi parametreleri veremez. Bu nedenle Şekil 3.18'de temsili olarak gösterilen yerel optimum yerine öğrenme sırasında tüm arama uzayında en az hata değerine ulaşılmalıdır. Yani en iyi parametre değerlerine karşılık gelen küresel optimuma erişilmesi gerekmektedir. Gong ve diğerleri Birlikte Evrim (BE) (Potter ve De Jong, 1994) yönteminin bu sorunu çözeceğini düşünerek GYBE_DSA yöntemini önermişlerdir. Çalışmalarında önerdikleri yöntemin başarısını analiz etmek amacıyla klasik Geri Yayımlı Derin Sinir Ağları (GY_DSA) yöntemini de kullanmışlardır. Bu kapsamda, el yazısıyla yazılmış 0'dan 9'kadar 28 x 28 piksellik toplam 60.000 resimden oluşan 10 sınıflı MNIST veri seti ile uçak, kuş, kedi gibi toplam 10 sınıfa sahip 32 x 32 piksellik 60.000 renkli görüntü içeren CIFAR-10 veri seti kullanılarak öğrenim gerçekleştirmişlerdir.

Değerlendirme sonucunda, GY_DSA yönteminin başarılı sonuçlar elde edemediği gözlenirken GYBE_DSA yöntemi içerdiği BE yöntemi ile takılma problemini aşarak çok başarılı bir öğrenim gerçekleştirebilmiştir. Bu nedenle yüksek doğruluğa sahip çatlak tespiti yapabilmek için GY_DSA yerine GYBE_DSA yöntemi çalışma içerisinde kullanılmıştır.



Şekil 3.18. Hata miktarı ve arama uzayına bağlı olarak yerel ve küresel optimumum temsili gösterimi

Kullanılan GYBE_DSA yönteminin çalışma mantığına bakıldığında, ilk başta GY ile öğrenme sırasında her döngüdeki parametrelerin hata değeri hesaplanır ve bir önceki hata değeriyle karşılaştırılır. Eğer hata değeri belirlenen değerden küçükse yerel optimuma takıldığı varsayılmaktadır. Dolayısıyla BE yöntemi bu durumu çözmek ve daha uygun parametre değerleri elde etmek için işleme dahil olur. Öncelikle GY algoritmasında en son kullanılan ağırlık ve yanlılık değerlerine sahip ağdaki her bir nöron alt görevlere ayrılır. Ancak ağ yapısında çok fazla nöron bulunması çok fazla alt görevin oluşmasına neden olmaktadır. Sonuç olarak her bir alt görev için uygun parametrelerin tespiti büyük bir işlem yüküne yol açacaktır. Bu durumu çözmek için sadece belirli alt görevlerin seçilmesi ve onlar üzerinde işlem yapılması daha verimli olacağı bilinmektedir (Gong ve diğerleri, 2021). Bu nedenle her bir alt görevin olgunluğu hesaplanır. Olgunluk bir alt görevin sahip olduğu ağırlık parametrelerin en uygun değerlere ne ölçüde ulaştığını temsil etmektedir. Dolayısıyla bir alt görevin olgunluğa ulaşması artık bu alt görevin ağırlık

değerlerinin istenilen değerlere eriştiğini göstermektedir. Bu nedenle en az olgunluğa ulaşmış yani en çok parametrelerini güncellemeye ihtiyacı olan alt görevler seçilmektedir.

Seçim işlemin ardından evrim süreci başlar. Evrim sürecinin başında her bir alt görev için rastgele popülasyon üretilir ve popülasyonlar içerisinde alt görevle aynı ağ yapısına sahip ama farklı ağırlık ve yanlılık değerleri bulunduran çeşitli alt görevler vardır. Ardından her bir alt görev için oluşturulan popülasyondaki tüm üyelerin hata değeri tek tek hesaplanır. En az hata değerine sahip bireyler seçilir ve bir sonraki nesle aktarılır. Aktarılma işleminden sonra bu bireylere mutasyon, çaprazlama gibi yöntemler uygulayarak popülasyonun geri kalan üyeleri üretilir. Bu işleme takiben tekrar hata değerleri hesaplanır ve en iyiler bir sonraki nesle aktarılır. Uygulanan işlem belirlenen nesil sayısına ulaşana kadar devam eder. İstenilen nesil sayısına ulaşıldığında popülasyondaki üyelerin hata değerleri hesaplanır ve her bir alt görevi temsil eden popülasyon içindeki sadece en az hata değerine sahip yani en iyi parametre değerlerine sahip alt görevler seçilir. Ardından tüm alt görevlerin parametreleri kullanılarak genel hata değeri hesaplanır. Eğer bulunan hata değeri BE yöntemine başlamadan önceki hata değerinde daha düşük çıkarsa daha uygun parametreler elde edildiği anlaşılır ve güncel parametreler artık yeni parametre değeri olarak kabul edilir. Son olarak bu değerler kullanılarak tekrar GY yöntemine geçiş yapılır.

Kısaca BE, GY ile öğrenme sırasında hata değerinde etkili bir şekilde iyileşme olmadığı zaman sisteme dahil edilir. BE çeşitli yöntemler uygulayarak en iyi parametre değerlerini ve buna bağlı olarak hata değerini hesaplar. Eğer daha iyi parametre değerlerine ulaşılırsa bu değerler yeni parametre değerleri olarak kabul edilir ve GY uygulamasına tekrar geçiş yapılır. Bu işlem istenilen hata değerine ulaşana kadar devam etmektedir. Uygulanan yöntemin açıklayan algoritma daha sonraki bölümlerde detaylı bir şekilde verilmiştir.

Son olarak bu çıktılar ile test verilerine ilerdeki bölümlerde açıklamaları yapılan Kesinlik, Duyarlılık ve F1-Score yöntemleri uygulanarak performans değerlendirilmesi yapılmış ve en yüksek başarıya sahip ağ yapısının ağırlık ve yanlılık terimleri çatlak tespitinde kullanmak için kaydedilmiştir.

3.3.3.1. Geri Yayılım

GY algoritması, DSA parametre öğrenimi için en yaygın kullanılan eğilim tabanlı bir eğitim algoritmasıdır ve zincir kuralı aracılığıyla DSA'da ki her parametreye göre hata fonksiyonunun kısmi türevini hesaplar. Daha sonra DSA'daki parametreler hesaplanan türevler kullanılarak güncellenir. Yapılan bu parametre güncelleme işlemi, bazı durdurma şartları karşılanana kadar tekrarlanır. GYBE_DSA yönteminde yerel optimum değere düşme ihtimalini düşürmek amacıyla GY algoritmasının düzenlileştirme terimi eklenmiş hali kullanılmıştır. GY'li parametre öğrenme algoritması için F(P), Denklem 3.4'te verilen hata fonksiyonuna bağlı olarak bulunur.

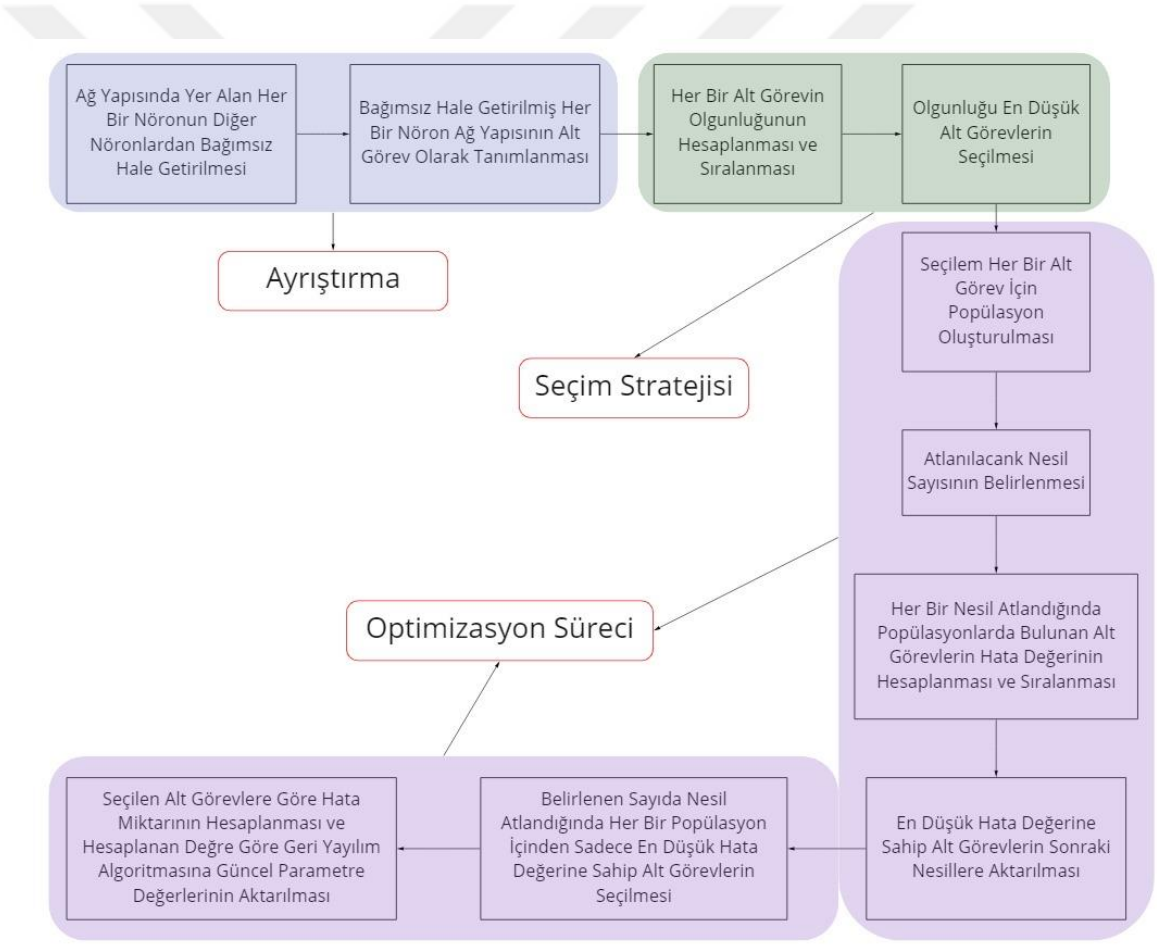
$$F(P) = \frac{2}{2|D|} \sum_{(x,y)\in D} \|o^L(x_k) - y\|_2^2 + \beta\|W\|_2 \tag{3.4}$$

Burada D eğitim verisini, x_k girdi değerlerini, y bilinen çıktı değerlerini, W bir DSA'daki tüm bağlantı ağırlıklarını, β ise düzenlileştirme terimini temsil etmektedir. Verilen denklem ile oluşturulan ağ yapısı ve ağırlık parametrelerinin hata miktarı bulunmaktadır ve hata miktarının değeri ne kadar küçükse o kadar iyi öğrenme yapıldığı anlamına gelir. Ancak hata miktarının çok küçükte olmaması gerekmektedir çünkü bu şekilde bir öğrenme yapıldığında sonucu ezberlemesi ve sadece o veriler üzerinde başarı elde edilebilmesine neden olmaktadır.

Diğer taraftan her veri seti ve her öğrenme için en iyi öğrenmeyi sağlayan sabit bir hata değeri bulunmamaktadır. Dolayısıyla öğrenme sırasında araştırmacı ya çeşitli sınırlayıcı hata değeri belirleyip en uygun hata değerini belirleyebilir ya da belirli bir hata değerine ulaşmak yerine belirli bir öğrenim devrine ulaşmayı tercih edebilir. Bu çalışma da çok fazla girdi değeri bulunduğu için sınırlayıcı hata değeri bulunamamış ve sadece sabit bir devir sayısı belirlenmiştir.

3.3.3.2. Birlikte Evrim

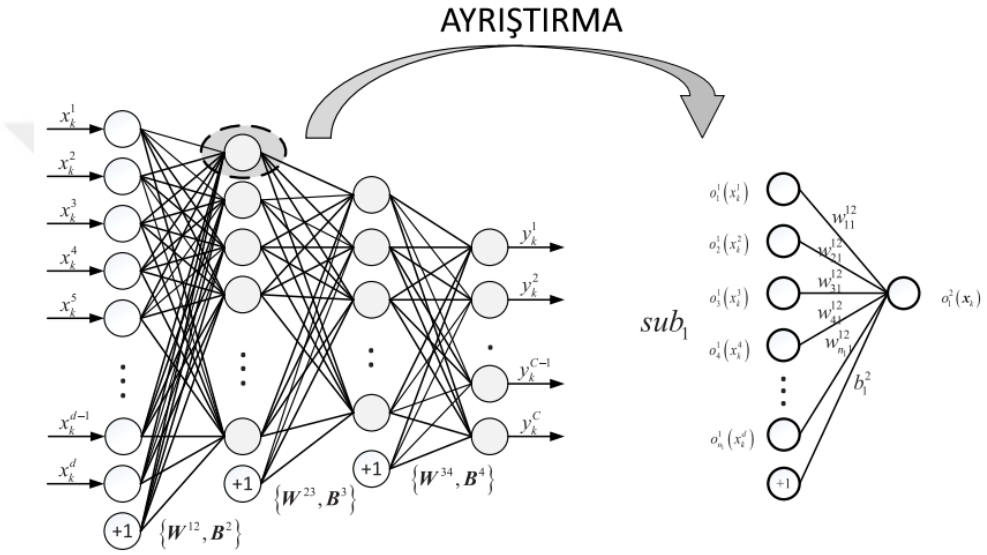
Birlikte Evrim (BE) algoritması ise bir tür Evrimsel Algoritma'dır (EA) (Xin Yao, 1999). EA'nın tamamı bir rastgele süreç üzerine kuruludur ve kısıtlı bilgi bulunan problemlere çözüm aramak için geliştirilmiştir. BE algoritması ağ yapısında yer alan tüm nöronları alt görevlere ayırır. Daha sonra, alt görevleri EA'larla ayrı ayrı geliştirir ve en uygun ağırlık parametreleri elde edilir. Bu durum, ilerleyen her süreçte arama alanını büyük ölçüde azaltır. BE tabanlı optimizasyon sistemi Ayrıştırma, Seçim Stratejisi, Optimizasyon Süreci olmak üzere 3 temel aşamadan oluşmaktadır ve Şekil 3.19'da akış diyagramı gösterilmektedir.



Şekil 3.19. Birlikte Evrim yönteminin akış diyagramı

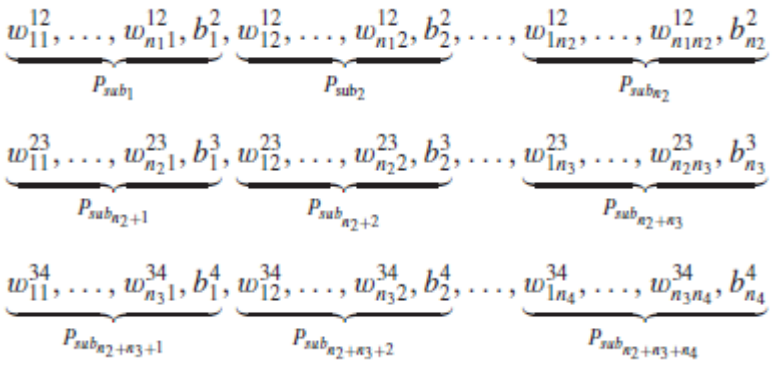
3.3.3.2.1. Ayrıştırma

Ayrıştırma metodu tam olarak sayısal bir yöntem değildir ve sadece nöronları ayrıştırma işlemidir. Bu yöntemde her bir ağdaki her bir nöron kendi ağındaki nöronlardan bağımsız hale getirilir böylece ileride anlatılacak olgunluk durumu bireysel olarak kontrol edilebilir. İşlemi daha iyi açıklamak adına Şekil 3.20'de birinci alt görevin temsili görüntüsü verilmiş ve varsayım yapılarak metot açıklanmıştır.



Şekil 3.20. Sinir ağlarının alt görevlere ayrıştırılmasının görünümü (Gong ve diğerleri, 2021)

Burada; bir giriş katmanı, bir çıkış katmanı, iki tanede gizli katmanı ile toplam 4 katmanı olan bir Çok Katmanlı İleri Beslemeli Ağ olduğu varsayılmıştır. Bu ağa Nöron Tabanlı Ayrıştırma (NTA) stratejisi uygulandığında ayrılma işlemi her bir nöron için yapıldığı için bu ağ toplam n2+n3+n4 (n1:1.katmanın nöron sayısı, n2:2. katmanın nöron sayısı, n3: 3.katmanın nöron sayısı, n4: 4.katmanın nöron sayısı) tane alt göreve ayrılır. Ayrılma işlemi sonucunda aşağıda verilen her bir alt görev $P_{sub_r} = \{W_{sub_r}, B_{sub_r}\}(\mathrm{r} = 1, \ldots, \mathrm{n2} + \mathrm{n3} + \mathrm{n4})$ olarak temsil edilir:



Burada, $w_{n_i n_j}^{xy}$ ağırlık parametresini, $b_{n_j}^{y}$ yanlılık terimini temsil etmektedir. Parametrelerin içerisinde bulunan x iki katman arasındaki bağlantıda sol taraftaki katmanı, y sağ taraftaki katmanı, n_i sol taraftaki nöronları, n_j ise sağ taraftaki nöronları ifade etmektedir. Sonuç olarak her bir alt görev iki katman arasındaki bağlantıda sol katmandaki nöron sayısı kadar yani sağ katmanda yer alan bir nöronun sol katmanda yer alan nöronlar arasındaki bağlantı sayısı kadar ağırlık parametresi içermektedir. Ayrıca bir ağ yapısında giriş katmanı hariç diğer katmanlarda yer alan tüm nöronların sayısı kadar alt görev bulunmaktadır.

3.3.3.2.2. Seçim Stratejisi

Öğrenme aşamasında bazı nöronlar yeterli seviyeye gelmesine rağmen öğrenmeye ve ağırlıklarını güncellemeye devam eder. Ancak bu durum hem gereksiz zaman kaybına hem de öğrenmenin başarısında probleme yol açacaktır. Bu nedenle belirli bir olgunluğa erişmiş nöronların öğrenmesinin durdurulması, hala olgunlaşmamış nöronların ise öğrenmesinin devam ettirilmesi performans açısından son derece verimli olacaktır. Bu durumdan yola çıkarak Denklem 3.5 ile her bir alt görev için olgunluk derecesi hesaplanır ve Denklem 3.6 ile Denklem 3.7 kullanılarak nöronun olgunluğu hesaplanır.

$$o_r{}^{l}(x_k) = A\left(\sum_{i=1}^{n} \mathrm{w}_{ir}^{(l-1)l}.\, o_i{}^{l-1}(x_k) + b_r^l\right) \tag{3.5}$$

$$G(s) = \begin{cases} 0, \alpha \ll s \ll 1 - \alpha \\ 1, diğer\ koşullarda \end{cases} \quad (3.6)$$

$$M_r = \sum_{k=1}^{N} G\left({o_r}^l(x_k)\right) \quad (3.7)$$

Burada x_k girdi değerlerini, w_{ir} ağırlık değerlerini, b_r^l girdi değerlerini, l katmanın numarasını, α ise eşik parametresini temsil etmektedir ve a bu çalışmada 0,3 olarak belirlenmiştir. Ancak bir alt görevi yeterli olgunluğa ulaştığını belirtmek için sabit bir olgunluk sınır değeri bulunmamaktadır. Sadece olgunluk değeri yüksek olan parametrelerin daha olgun olduğu yani daha iyi ağırlık parametrelerine sahip olduğu bilinebilmektedir. Bu nedenle seçim aşamasında tüm alt görevlerin olgunlukları hesaplanıp sıralanmış ve en düşük olgunluk değerine sahip olan yani ağırlık parametrelerini güncellemeye ihtiyacı olan alt görevler seçilmektedir.

3.3.3.2.3. Optimizasyon Süreci

Optimizasyonu sağlamak için seçilen her bir alt görevin en uygun parametre değerlerine sahip olması gerekir. Bu gerekliliği sağlamak için uygun alt görev popülasyonuna yani çeşitli alt görevler içeren bir topluluğa gerek vardır. Bu popülasyonu belirlemek içinde Diferansiyel Algoritmanın yüksek performansa sahip yöntemlerinden biri olan Kendi Kendine Uyarlanabilir Diferansiyel Evrim (KUDE) (Qin, Huang ve Suganthan, 2009) yöntemi kullanılmaktadır. KUDE, birbiriyle bağlantılı verilerin bulunduğu problemlerde etkili bir şekilde çalışabilen ve içerdiği operatörler nedeniyle popülasyon tabanlı sezgisel optimizasyon yöntemidir. Yöntem içerisinde yer alan çeşitli yöntemler uygulanarak en uygun popülasyon verileri seçilmektedir.

KUDE algoritması ile uygun alt görevler seçildikten sonra Denklem 3.8 kullanılarak her bir alt görevin hata değeri hesaplanırken Denklem 3.9 ile popülasyon sayısı kadar birey içeren bir popülasyon oluşturulur.

$$F_s(P_{sub_r}) = \frac{1}{\frac{2}{2|D_s|}\sum_{(x,y)\in D_s}\|o^L(x_k) - y\|_2^2 + \beta\left\|W_{sub_r}\right\|_2 + 10^{-8} + \sigma_s M_r} \quad (3.8)$$

$$P_{sub_r}^{i_p}(j) = \begin{cases} P_{sub_r}^*(j).rand(0,2), & i_p = 1, \dots, NP-1 \\ P_{sub_r}^*(j), & i_p = NP \end{cases} \quad (3.9)$$

Burada D_s eğitim verisinden rastgele seçilmiş küçük alt kümeyi, σ_s ağırlık parametrelerini, NP ise popülasyon boyutunu temsil etmektedir ve NP değeri 50 olarak belirlenmiştir. Denklem 3.9 kullanılarak seçilen her bir alt göreve ait bir popülasyon yani benzer yapıda ama farklı ağırlık parametreleri bulunduran farklı alt görevler oluşturulmakta ve en iyi ağırlık parametrelerine sahip ağ parametrelerini belirleyebilmek için her bir popülasyon bir döngüye sokularak her döngünün sonunda en iyi alt görevler dikkate alınarak yeni bir popülasyon oluşturulmaktadır. En iyi alt görevleri belirlemek içinde her bir alt görevin hata değeri bulabilen Denklem 3.8 kullanılmaktadır. Dolayısıyla bir popülasyonda en az hata değerine sahip olan alt görev en iyi olarak belirlenir bu duruma göre yeni nesiller oluşturulur.

3.3.3.3. Geri Yayılımlı Birlikte Evrim Algoritması

DSA'lardaki parametreleri öğrenmek için kullanılan GY tabanlı eğilim azalmasının ve BE tabanlı optimizasyonun iyi ve kötü olduğu noktalar vardır. GY, iyi hesaplama verimliliğine sahiptir, ancak başlangıç hassasiyeti ve düşük yerel optimuma sıkışma eğilimindedir. BE optimizasyonu ise eğilim içermez ve bu nedenle, genellikle hesaplama açısından uzun süreler harcadığı için verimli değildir ancak GY'nin yaşadığı sorunları önleyebilir. Bu nedenle Gong ve diğerleri (2021) tarafından BE'yi GY ile birleştirilmesi önerilmiş böylece bu iki öğrenme tekniği birbirini tamamlayarak öğrenme performansı önemli oranda iyileştirilmektedir. Yöntemin algoritmasında kullanılan simgelerin açıklamaları Şekil 3.21'de sunulurken yöntemin algoritması Şekil 3.22'de verilmiştir. Ayrıca Gong ve diğerleri tarafından çok paradigmalı sayısal hesaplama yazılımı olan MATLAB programlama dilinde geliştirilen GYBE_DSA yöntemi, görüntü işleme tekniklerini uygulamak için daha kullanışlı olan PHYTON programlama dili içerisinde tekrar yazılmış ve yazılım kodları EK-1'de gösterilmiştir.

D: Geri Yayılım için belirlenen döngü sayısı

$H_{(t)}$: Mevcut döngüde hesaplanan hata değeri

$H_{(t-1)}$: Bir önceki döngüde hesaplanan hata değeri

$H_{(BE)}$: Birlikte Evrim yaklaşımıyla elde edilmiş parametre verileri kullanarak hesaplanmış hata değeri

YOS: Öğrenmenin yerel optimuma takıldığını kontrol eden sınır değeri

NS: Her bir alt görevin ilerleyeceği nesil sayısı

SAS: Seçilmiş alt görev sayısı

$P_{(t)}$: Mevcut döngüdeki parametreler

$P_{(BE)}$: Birlikte Evrim yoluyla elde edilmiş parametreler

Şekil 3.21. GYBE_DSA yönteminin algoritmasında kullanılan simgeler ve açıklamaları

```
Rastgele olarak DSA parametrelerini belirle
for t = 1, … , D:
    if t = 1:
        Denklem 3.3 kullanarak hata değeri hesapla
        Hata değeri kullanarak parametre değerlerini güncelle
    else:
        Denklem 3.3 kullanarak hata değeri hesapla
        Mevcut döngüde hesaplanan hata değerini bir önceki döngüde hesaplanan değerinden çıkar
        if H(t) - H(t-1) < YOS:
            Ağ yapısı ayrıştırılmaya uğrat ve alt görevler elde et
            Her bir alt görevin olgunluğu Denklem 3.6 ile hesapla
            Olgunluğa göre sıralama yap ve en az olgunlaşmış alt görevleri seç
            for k = 1, … , NS:
                for j = 1, … , SAS:
                    if k = 1:
                        Denklem 3.8'' kullanarak popülasyon oluştur
                    Denklem 3.7'yi kullanarak popülasyondaki her bir alt görevin hata değerini hesapla
                    En az hata değerine sahip olan alt görevleri bir sonraki nesile aktar
                    Aktarılan alt görevlere mutasyon, çaprazlama gibi mekanizmalar uygula ve yeni bir popülasyon oluştur
            Her bir alt görevin yer aldığı popülasyondaki alt görevlerin hatalarını Denklem 3.7 kullanarak hesapla
            Her bir popülasyondaki sadece en az hata değerine sahip sadece en iyi alt görevi seç
            Denklem 3.3 ve tüm seçilmiş alt görevler kullanarak hata değerini hesapla
            if H(t) > H(BE):
                P(t) = P(BE)
```

Şekil 3.22. GYBE_DSA yönteminin algoritması

3.3.3.4. Kesinlik, Duyarlılık ve F1-Score

Öğrenmenin sonunda, GYBE_DSA kullanılarak elde edilen çıktı verilerinin mevcut veriler ile karşılaştırarak önerilen yöntemlerin performansının belirlenmesi gereklidir. Bu gerekliliği sağlamak içinde genel olarak Kesinlik, Duyarlılık ve F1-Score yöntemleri uygulanmaktadır. Kesinlik ve Duyarlılık temel olarak Pozitif olarak tahmin edilen değerlerin gerçekten kaç adedinin Pozitif olduğunu Denklem 3.10 ve Denklem 3.11 kullanarak hesaplayan yöntemlerdir. F1-Score ise Kesinlik ve Duyarlılık değerlerinin harmonik ortalamasını alarak daha güvenilir bir başarı oranı veren bir yöntemdir ve Denklem 3.12 ile hesaplanmaktadır.

$$Kesinlik = \frac{TP}{TP + FP} \tag{3.10}$$

$$Duyarlılık = \frac{TP}{TP + FN} \tag{3.11}$$

$$F1 - Score = \frac{2 * (Kesinlik * Duyarlılık)}{(Kesinlik + Duyarlılık)} \tag{3.12}$$

Burada TP çatlak tespiti yapılan bölgede çatlak olduğu tahmin edilen ve gerçekte çatlak bulunduran, FP çatlak tespiti yapılan bölgede çatlak olduğu tahmin edilen ama gerçekte çatlak bulundurmayan, FN ise çatlak tespiti yapılan bölgede çatlak olmadığı tahmin edilen ve gerçekte de çatlak bulundurmayan tüm test örneklerinin sayısıdır. Ayrıca üç yöntemin PHYTON yazılım kodları EK-2'de verilmektedir.

Temel olarak bu değerler 0 ile 1 arasında sonuç değerleri vermektedir. Elde edilen değerin 1 olması hatasız bir şekilde tespit yapıldığını göstermektedir. Ancak böyle bu durum söz konusu değildir ve her bir başarı değerlendirmesinde iyi bir tespit sistemi tasarlamak için olabildiğince 1 değerine yaklaşmak gerekmektedir.

3.3.4. Görüntü İşleme ile Çatlak Bölgesinin Kabaca Belirlenmesi

Çatlak tespiti aşamasında, zaman kazanmak amacıyla GYBE_DSA yöntemi ile öğrenilmiş veriler önemli bir yer tutmaktadır. Bu nedenle çeşitli ağ yapıları uygulanarak elde edilen öğrenme süreçlerinden en yüksek başarı oranına sahip ağ yapısına ait verilerin kullanılması çatlak tespitinin daha başarılı olmasını sağlamaktadır. Bu durum göz önünde tutularak ağ yapılarına göre optimum sonuç verileri daha sonra çatlak tespitinde kullanılması için dosya olarak kaydedilir.

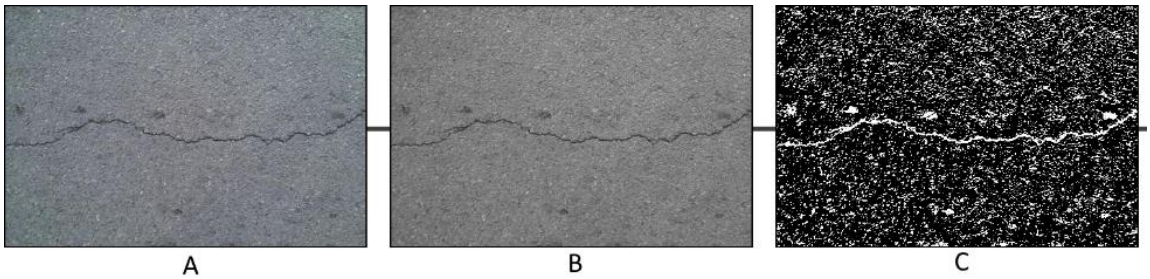
GYBE_DSA yöntemi ile sadece 80 x 80 piksel boyutlarındaki alanlarda çatlak tespiti yapılabilmektedir ama görsel içerisinde tam olarak nerede GYBE_DSA yöntemi ile öğrenilmiş ağırlık verileri kullanılarak çatlak tespiti yapılacağı bilinmemektedir. Dolayısıyla öncelikle çatlak tespiti yapılacak bölgenin kabaca konumu hızlı bir şekilde belirlenmelidir. Bu amaca ulaşmak içinde en etkili yol görüntü işleme tekniklerini kullanmaktır.

Görüntü işleme ile çatlak tespitinde Şekil 3.9'da gösterildiği üzere temel olarak 5 aşama bulunmaktadır. İlk aşamada görsele eşikleme yöntemi uygulanmakta, 2.Aşamada görsel içerisinde yer alan çok küçük nesneler silinmekte çok küçük boşluklar doldurulmaktadır. Sonraki aşamada küçük boyutlu çatlak olmayan nesneler görsel içerisinden arındırılmakta ve 4.aşamaya geçiş yapılmaktadır. Bu aşamada birbirine yakın olan çatlak nesneleri birleştirilmekte son aşamada ise görsel içerisinden daha önce silinemeyen büyük boyutlu çatlak olmayan nesnelerden kurtulmaktadır. Sonuç olarak görsel içerisinde kabaca sadece çatlağın yer aldığı bir görsel elde edilmektedir.

3.3.4.1. Uyarlanabilir Eşikleme Yönteminin Uygulanması

Eşikleme yöntemi gri renkli bir görseli siyah-beyaz bir görsele çeviren bir metottur ve farklı ortam koşulları için en uygunu Uyarlanabilir Eşikleme Yöntemi'dir. Ancak eşikleme yöntemleri sadece gri renkteki görsellere uygulanabilmektedir. Bu nedenle öncelikle Şekil 3.23A'da örnek olarak gösterilen renkli görseller Şekil 3.23B'de gösterildiği gri renge çevrilmelidir. Bu dönüşümü sağlamak için Denklem 3.3

kullanılmıştır. Gri renkli görseller elde edildikten sonra ise görsele Uyarlanabilir Eşikleme Yöntemi uygulanmış Şekil 3.23C'de gösterilen siyah-beyaz renkte görsel elde edilmiştir.



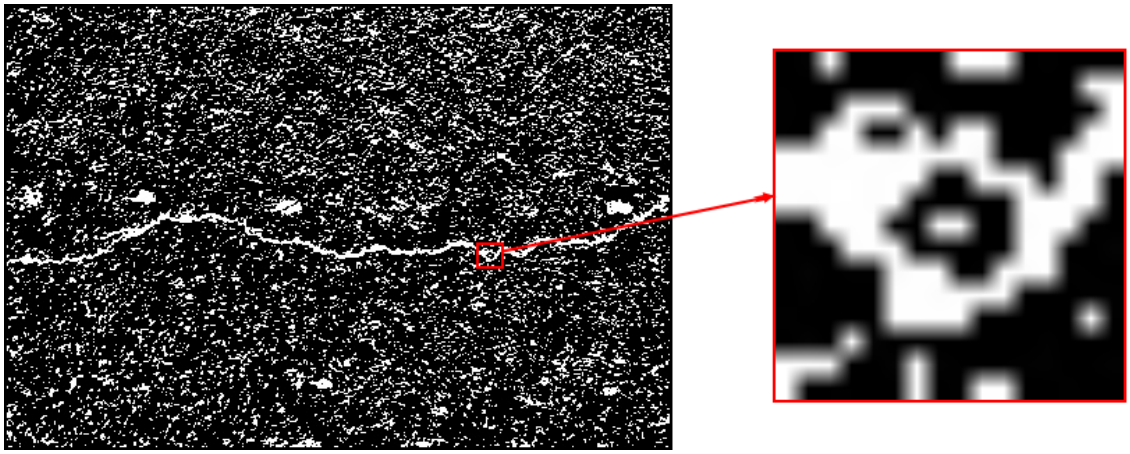
Şekil 3.23. Görselin gri renge çevrilip eşikleme yönteminin uygulanması
A) Orijinal görsel **B)** Gri hale çevrilmiş görsel **C)** Eşikleme yöntemi uygulanmış görsel

3.3.4.2. Çok Küçük Nesne ve Boşlukların Giderilmesi

Eşikleme yöntemi uygulandıktan sonra görsel içerisinde çatlak belli olsa da görsel içerisinde çatlak olmayan başka nesneler vardır. Bu nedenle ilk başta beyaz renkli bölgelerin çatlak nesnelerini siyah renkli bölgelerin ise çatlak olmayan nesneleri temsil ettiği kabulü yapılmıştır. Şekil 3.23C'de gözüktüğü üzere görsel içerisinde çatlak olmayan fazla sayıda çok küçük nesneler bulunmaktadır ve çatlak alanını belirlemek için bu nesnelerden kurtulmak gerekmektedir.

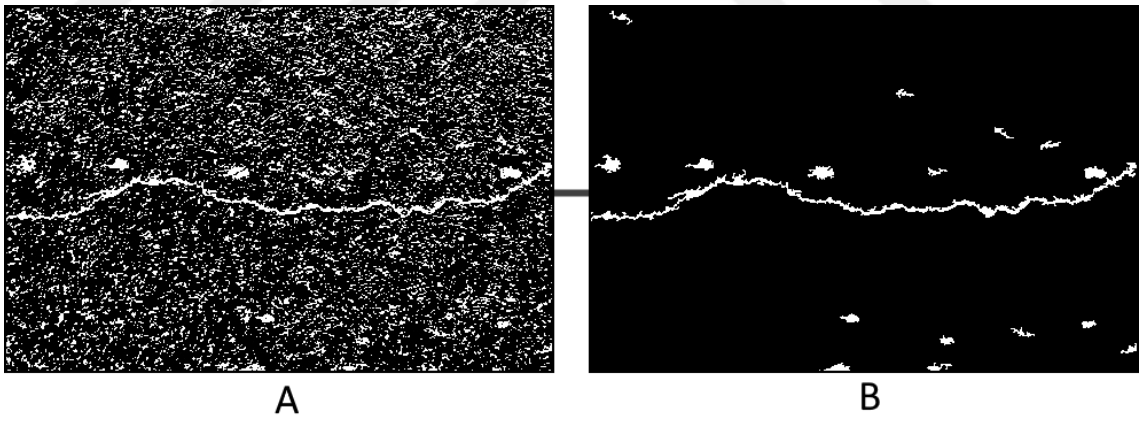
Bu amaçla görüntü işleme ilgili çeşitli algoritmalar içeren "scikit-image" kütüphanesinde bulunan "remove_small_objects" yönteminden yararlanılmıştır. Bu yöntem temel olarak görsel içerisindeki her bir nesnenin kaç piksel değerinden oluştuğunu hesaplamakta ve belirli değerden küçük olan değerleri tespit edip onları görsel üzerinden silmekte yani siyah renge çevirmektedir.

Diğer yandan Eşikleme Yöntemi sonucunda Şekil 3.24'te gösterilen görselin çatlak bölümüne yakınlaştırılmış ve çatlak içerisinde küçük boşluklar olduğu görülmüştür. Çatlak nesnelerinin içinde boşlukların bulunması çatlağın algılanmasında problem yaratabileceği için boşluklar doldurulmalıdır. Bu nedenle yine "scikit-image" kütüphanesinde bulunan "remove_small_holes" yönteminden yararlanılmıştır.



Şekil 3.24. Görseldeki çatlak bölgesinin içinde yer alan çok küçük boşluk

Sonuç olarak Şekil 3.25A'da gösterilen eşikleme yöntemi uygulanmış görsele "scikit-image" kütüphanesinin "remove_small_objects" ve "remove_small_holes" yöntemleri uygulanarak Şekil 3.25B'de gösterilen çok küçük nesne ve boşluklardan kurtarılmış görsel elde edilmiştir.



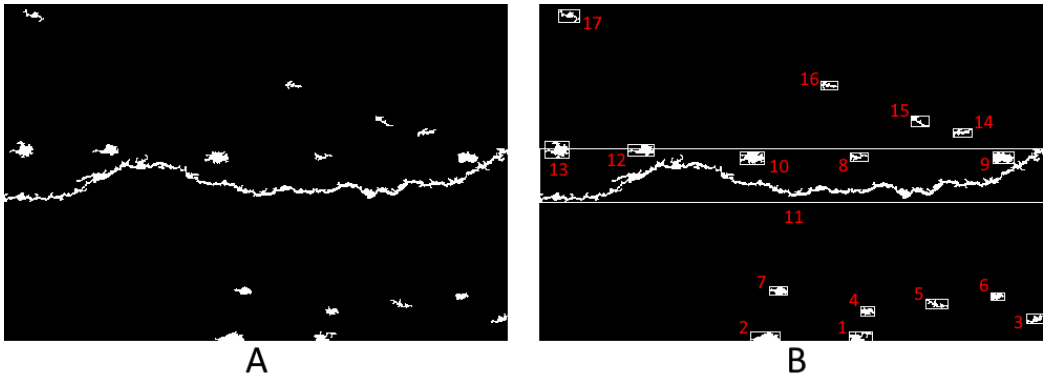
Şekil 3.25. Görsel içerisindeki çok küçük nesne ve boşluklardan kurtulmak için "scikit-image" kütüphanesinin kullanılması

A) Eşikleme yöntemi uygulanmış görsel **B)** Çok küçük nesne ve boşluklardan arındırılmış görsel

3.3.4.3. Küçük Boyutlu Nesnelerin Arındırılması

Çatlak tespiti yapılmak istenen görsel içerisinde temizleme işlemi uygulanmasına rağmen içerisinde hala çatlak olmayan ama çatlak olarak temsil edilen nesneler bulunma ihtimali yüksektir ve bu durum tespit aşamasında çeşitli problemlere yol açabilir. Bu nesneleri görsel içerisinden kaldırmak için çatlağın geometrik özelliklerinden yararlanılması sorunu çözebilir ve bu amaçla resim içindeki nesnelerin özelliklerinin bulunması için OpenCV kütüphanesinde yer alan "findCountours" yöntemi kullanılmaktadır. Bu yöntem temel olarak siyah beyaz bir görsel içerisindeki beyaz renkte birbirinden bağımsız olan şekilleri bir nesne olarak algılamakta ve her bir nesne içerisinde yer alan her bir pikselin konumları elde edilmektedir. Elde edilen konumlar kullanılarak görsel verisi içinde bulunan tüm birbirinden bağımsız nesnelerin önemli geometrik özelliklerine erişilmekte ve bu özelliklerinden yararlanarak çatlak olma ihtimali olmayan nesneler görsel içerisinde silinebilmektedir.

Silme işleminde ilk başta çalışmamızda kullandığımız CFD veri setinde yer alan 10 farklı görselden elde edilen çatlak ve çatlak olmayan nesnelerin geometrik özellikleri 2 farklı yöntem uygulanarak tespit edilmiştir. İlk yöntemde örnek olarak Şekil 3.26A'da gösterilen görsel içerisinde yer alan numaralandırılmış her bir nesneyi kapsayan ve temsil eden dik açılı bir dikdörtgen Şekil 3.26B'de gösterildiği üzere belirlenmiş ve ardından Şekil 3.26B'de belirlenen görselde numaralandırılmış her bir nesne için çizilmiş dikdörtgenlerin kenar uzunlukları, alan oranı, eğimi, doluluk oranı ve çatlak içeriği Çizelge 3.3'te örnek gösterildiği gibi kaydedilmiştir. Temel olarak bu işlem CFD veri seti içerisinde yer alan sadece 10 görsel için uygulanmış diğer görseller yöntemin sonunda başarıyı test etmek için kullanılmıştır.



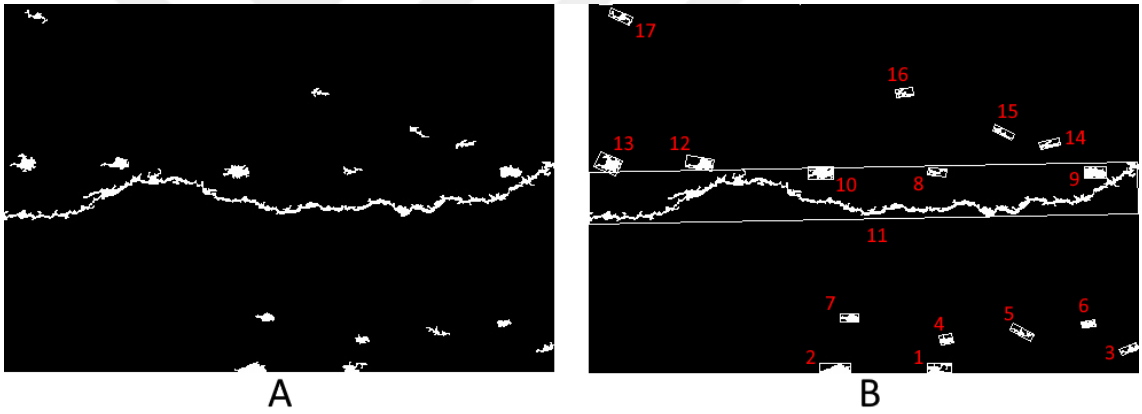
Şekil 3.26. Görselde bulunan nesnelerin dik açılı dikdörtgen alan yöntemi uygulanarak dikdörtgenler ile temsil edilmesi
A) Ön aşamadaki yöntemleri uygulanmış görsel **B)** Dik açılı dikdörtgen alan yöntemi uygulanması sonucu nesneleri kapsayan dik açılı dikdörtgenleri çizilmiş görsel

Çizelge 3.3. Dik açılı dikdörtgen alan yöntemi uygulanmasıyla öğrenilen nesnelerin geometrik özellikleri

Normal dikdörtgen alan yöntemiyle belirlenen özellikler						
Nesne Numarası	**Büyük Kenar (piksel)**	**Küçük Kenar (piksel)**	**Alan Oranı**	**Eğim**	**Doluluk Oranı**	**Çatlak İçeriyor mu?**
1	22	9	0,00129	0,409	0,470	Hayır
2	28	9	0,00164	0,321	0,567	Hayır
3	16	9	0,00094	0,563	0,410	Hayır
4	13	9	0,00076	0,692	0,513	Hayır
5	21	9	0,00123	0,429	0,302	Hayır
6	13	7	0,00059	0,538	0,637	Hayır
7	17	8	0,00089	0,471	0,574	Hayır
8	17	8	0,00089	0,471	0,382	Hayır
9	20	11	0,00143	0,550	0,682	Hayır
10	23	12	0,00180	0,522	0,616	Hayır
11	480	51	0,15938	0,106	0,106	Evet
12	25	11	0,00179	0,440	0,462	Hayır
13	23	16	0,00240	0,696	0,438	Hayır
14	18	8	0,00094	0,444	0,438	Hayır
15	17	10	0,00111	0,588	0,294	Hayır
16	16	8	0,00083	0,500	0,391	Hayır
17	20	12	0,00156	0,600	0,271	Hayır

Burada küçük kenar ve büyük kenar Şekil 3.26B'de gösterilen her bir nesne için belirlenen dikdörtgen alanın küçük ve büyük kenarlarıdır. Alan oran değeri belirlenen dikdörtgenin alanının ana görselin alanına oranlanmasıyla elde edilirken eğim değeri küçük kenarın büyük kenara bölümlenmesiyle bulunmuştur. Doluluk oranı ise belirlenen dikdörtgen alan içerisinde bulunan nesnenin alanının dikdörtgenin alanıyla oranlayarak bulunmuştur.

İkinci yöntemde ise örnek olarak Şekil 3.27A'da gösterilen görsel içerisinde yer alan numaralandırılmış her bir nesneyi kapsayan ve temsil eden en küçük dikdörtgen Şekil 3.27B'de gözüktüğü gibi temsili olarak belirlenmiş ve ardından Şekil 3.27B'de belirlenen görselde numaralandırılmış her bir nesne için çizilmiş dikdörtgenlerin kenar uzunlukları, alan oranı, eğimi ve çatlak içeriği Çizelge 3.4'te gösterildiği gibi kaydedilmiştir.



Şekil 3.27. Görselde bulunan nesnelerin en küçük dikdörtgen alan yöntemi uygulanarak dikdörtgenler ile temsil edilmesi
A) Ön aşamadaki yöntemleri uygulanmış görsel **B)** En küçük dikdörtgen alan yöntemi uygulanması sonucu nesneleri kapsayan dik açılı dikdörtgenleri çizilmiş görsel

Çizelge 3.4. En küçük dikdörtgen alan yöntemi uygulanmasıyla öğrenilen nesnelerin geometrik özellikleri

En küçük alan yöntemiyle belirlenen özellikler					
Nesne Numarası	**Büyük Kenar (piksel)**	**Küçük Kenar (piksel)**	**Alan Oranı**	**Eğim**	**Çatlak İçeriyor mu?**
1	21	8	0,00109	0,381	Hayır
2	27	8	0,00141	0,296	Hayır
3	16	6	0,00061	0,361	Hayır
4	11	8	0,00060	0,738	Hayır
5	21	7	0,00090	0,326	Hayır
6	12	5	0,00044	0,423	Hayır
7	16	7	0,00073	0,438	Hayır
8	16	7	0,00069	0,439	Hayır
9	19	10	0,00124	0,526	Hayır
10	22	11	0,00158	0,500	Hayır
11	479	45	0,14021	0,094	Evet
12	23	10	0,00157	0,437	Hayır
13	21	14	0,00192	0,695	Hayır
14	17	6	0,00072	0,362	Hayır
15	18	6	0,00073	0,326	Hayır
16	15	7	0,00070	0,467	Hayır
17	20	8	0,00097	0,387	Hayır

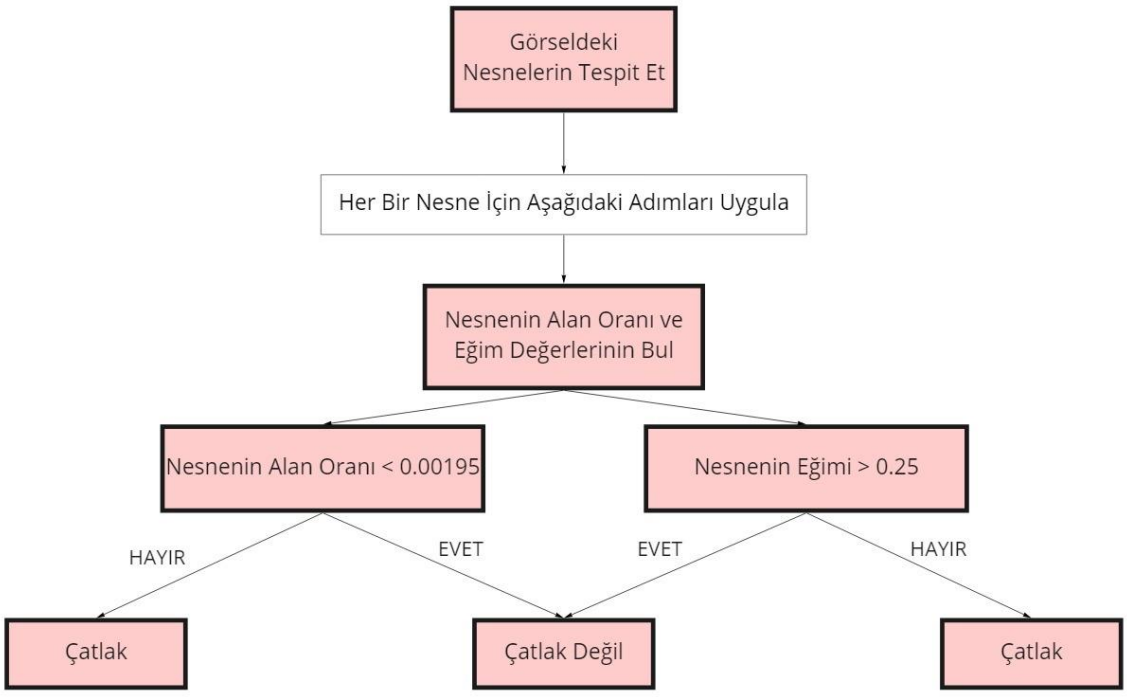
Burada küçük kenar ve büyük kenar Şekil 3.27B'de gösterilen her bir nesne için belirlenen dikdörtgen alanın küçük ve büyük kenarlarıdır. Alan oranı değeri dikdörtgenin alanının ana görselin alanına oranlanmasıyla elde edilen değerdir ve eğim değeri ise küçük kenarın büyük kenara bölümlenmesiyle bulunmuştur.

Son olarak çatlak olup olmadığı bilinen nesnelerin belirlenen özellik değerleri kullanılarak farklı görsellerde bir nesnenin çatlak olup olmadığını ayırt edebilecek sınır değerlerinin hesaplanması gerekmektedir. Bu amaçla tüm elde edilen veriler Excel formatında bir araya getirilmiş ve her bir özellik için çeşitli değerler verilerek en çok ayırt edici özellikleri ve değerleri bulunmaya çalışılmıştır. İlk başta rastgele olarak her bir özellik için sınır değerleri belirlenmiş ve bu sınır değerleri içerisinde olan değerlerin

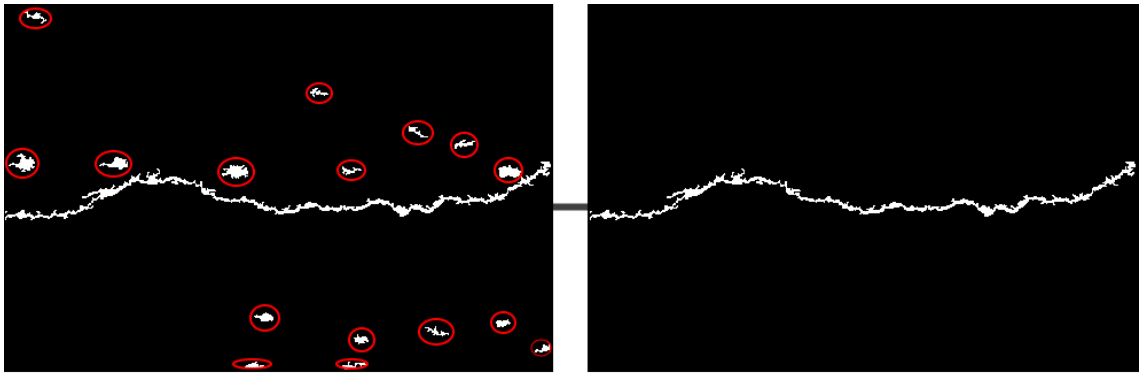
çatlak olmayan nesneler olarak tanımlanırken içerisinde olmayan nesneler çatlak nesneleri olarak belirlenmiştir.

Belirleme işleminden sonra nesnelerin gerçekte çatlak içerip içermediği karşılaştırılmıştır. Ardından çatlak olarak tespit edilmesi gereken ama çatlak olmayan nesne olarak tespit edilen nesnelerin sayısını minimumda tutmak üzere sınır değerleri rastgele olarak sürekli değiştirilmiştir.

Yapılan denemeler sonucunda bir nesnenin çatlak olup olmadığını dik açılı dikdörtgen alan yöntemiyle hesaplanan alan oranı değeri ile en küçük dikdörtgen alan yöntemiyle belirlenen eğim özelliklerinin en iyi şekilde ayırt edebileceği görülmüştür. Ayrıca bu iki özellik için çeşitli sınır değerleri belirlenmiş ve en uygun ayırt eden sınır değerleri bulunmuştur. Sonuç olarak Şekil 3.28'de verilen akış diyagramı kullanılarak Şekil 3.29A'da örnek olarak verilen bir görselde küçük boyutlu çatlak olmayan nesneler Şekil 3.29B'de gösterildiği üzere belirlenebilmekte ve resim içerisinden silinebilmektedir.



Şekil 3.28. Görsel içerisinde bulunan küçük boyutlu nesnelerin görsel içerisinden silinmesini sağlayan akış diyagramı

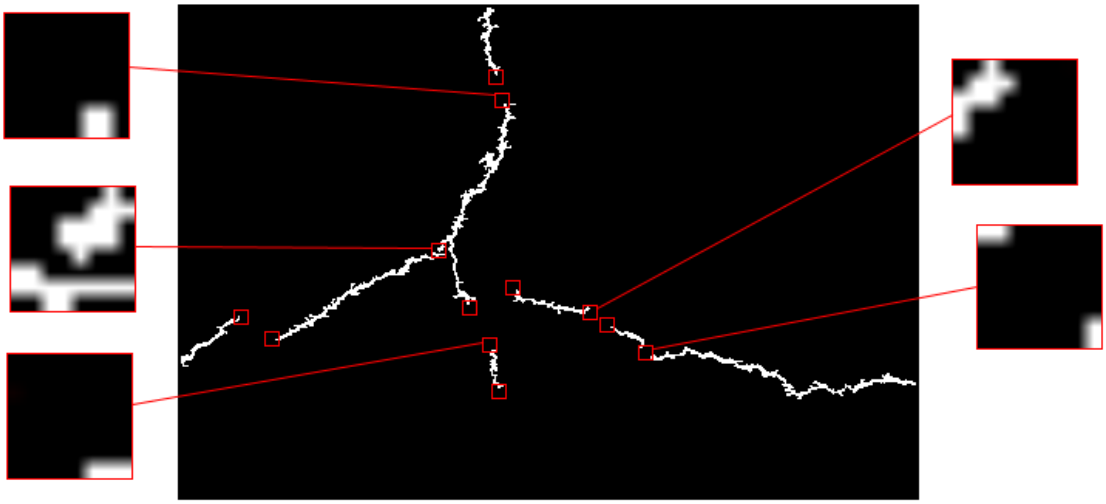


Şekil 3.29. Görsel içerisinde bulunan belirli boyuttaki çatlak olmayan nesnelerin silinmesi
A) Eşikleme yöntemi uygulanmış ve çok küçük nesne ve boşluklardan kurtulmuş görsel
B) Çatlak olmayan nesnelerden arındırılmış görsel

3.3.4.4. Nesnelerin Birleştirilmesi

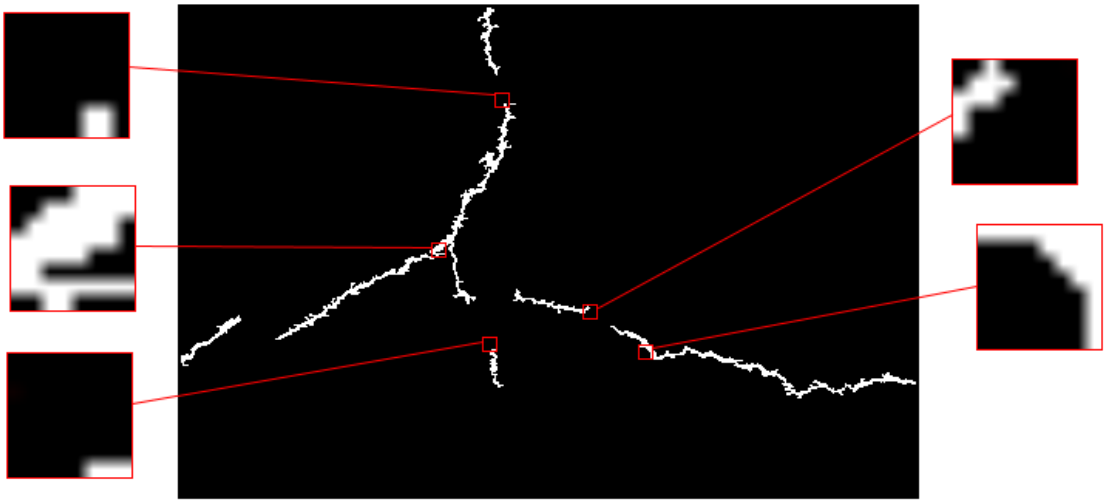
Görsellerde yer alan çatlaklarda çatlak boydan boya uzandığı gözle görülmesine rağmen bilgisayar ortamında çatlak aralarındaki boşluklar tespit edilebilmektedir ve bu durum çatlakların birbiriyle bağlantısının olmamasına ve farklı nesneler olarak algılanmasına neden olmaktadır. Dolayısıyla çatlak tespiti yapılırken çatlak nesneleri içerisinde ortak bölgelerin olması durumunda bile farklı nesneler olarak algılandıkları için bu bölgeler tekrar tekrar taranmaktır ve bu durum zaman açısından probleme neden olmaktadır. Bu durumu ortadan kaldırmak ve tespit aşamasındaki hızı ve doğruluğu artırmak için bu nesneler birleştirilmesi uygun görülmüştür.

Bu aşamada birleştirme işlemi gerçekleştirmek için öncelikle görsel içerisindeki her bir çatlak nesnesi tespit edilir. Ardından her bir nesnenin alınabilecek uç noktalarında Şekil 3.30'da örnek gösterildiği gibi 10 x 10 piksel boyutlarında olacak şekilde bölgeler alınır ve bu bölgeler içerisinde nesne taraması yapılır.



Şekil 3.30. Görselde bulunan her bir nesnenin alınabilecek uç noktalarından bölgeler alınması

Tarama sonucunda bölgeler içerisinde nesne sayısı belirlenir ve bölgeler içinde sadece bir tane nesne bulunursa nesnenin bu bölgesinde başka bir çatlak nesnesi olmadığı anlaşılır. Ancak birden fazla nesne bulursa bulunduğu bölgede Uyarlanabilir Eşikleme Yöntemi sonucu iki parçaya ayrılmış çatlak nesnesi olduğu tespit edilir ve Şekil 3.31'de gösterildiği üzere bu iki birbirinden ayrılmış nesneler birleştirilir.

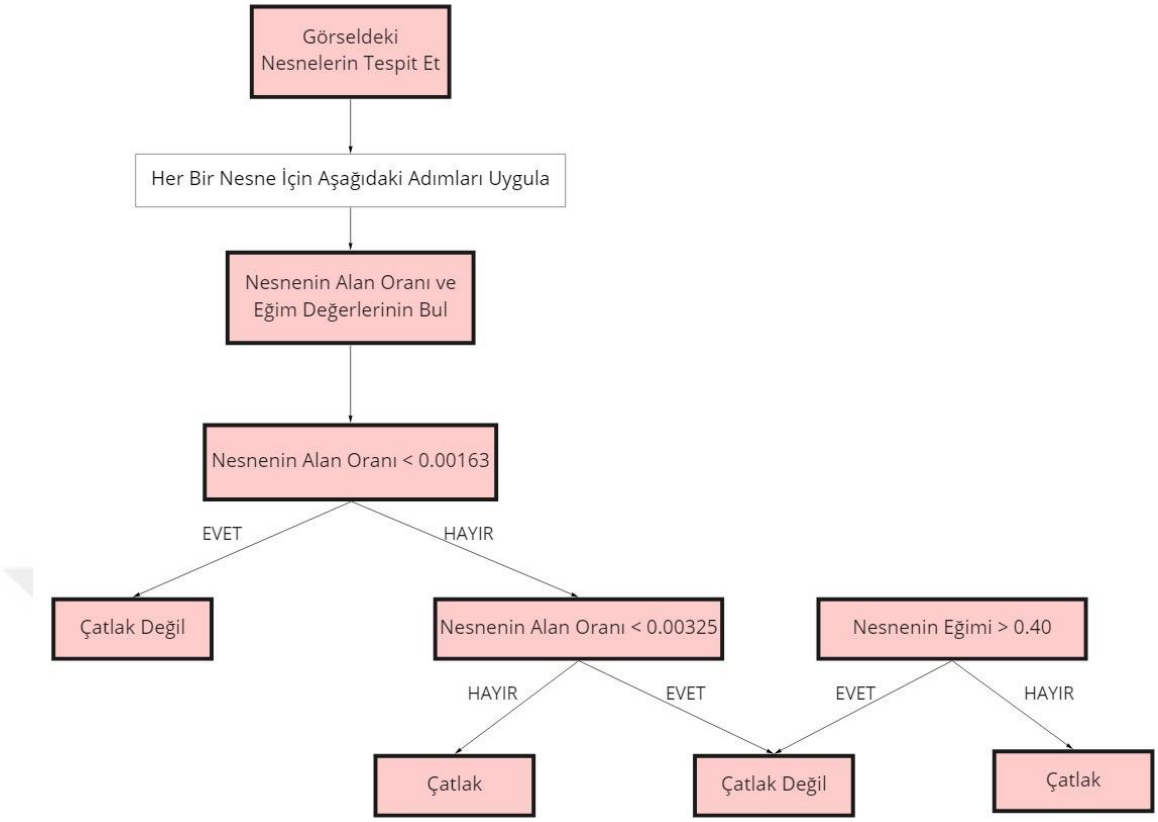


Şekil 3.31. Görselde bulunan ve birbirine yakın olan iki parçaya ayrılmış çatlak nesnelerinin birleştirilmesi

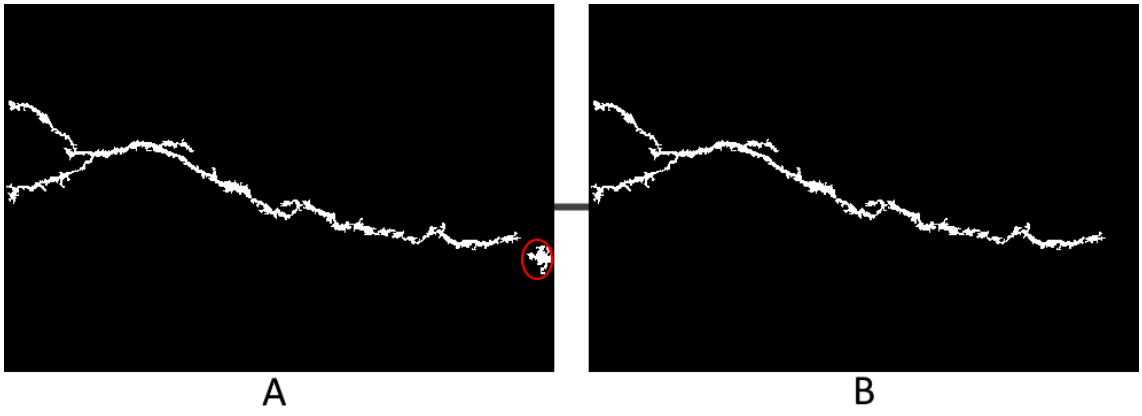
3.3.4.5. Büyük Nesnelerin Temizlenmesi

Çatlak tespiti yapılacak görsellerde genellikle küçük boyutlarda çatlak olmayan nesneler tespit edilmektedir ve bunlar önceki aşamalarda kullanılan yöntemler yardımıyla temizlenmektedir. Ancak bu yöntemler ile sadece küçük boyuttaki nesneleri ortadan kaldırılmaktadır ama nispeten daha büyük olan nesneler olduğu gibi durmaktadır ve bu boyutlara yakın boyuttaki çatlak nesneleri birbiriyle birleştiği için bu temizleme işlemi bunlar için problem oluşturmamaktadır. Bu ayrımı yapmak için "Küçük Boyutlu Nesnelerin Arındırılması" bölümde bahsedilen nesnelerin geometrik özelliklerinden yararlanılmıştır. Yine çeşitli denemeler yapılarak ilk başta en ayırt edici özellikler belirlenmiş ardından en uygun ayırt eden özellik değerleri bulunmuştur.

Denemeler sonucunda bir nesnenin çatlak içerip içermediğini en iyi şekilde ayırt eden özelliklerinin en küçük dikdörtgen alan yöntemiyle belirlenen alan oranı değeri ile eğim değeri olduğu bulunmuştur. Ayrıca en iyi ayırt eden değerler belirlenmiş ve sonuç olarak Şekil 3.32'de verilen akış şeması izlenerek Şekil 3.33A'da örnek olarak gösterilen bir resimde büyük boyutlu çatlak olmayan nesneler Şekil 3.33B'de gösterildiği gibi belirlenebilmekte ve görsel içerisinden temizlenebilmektedir.



Şekil 3.32. Görsel içerisinde bulunan büyük boyutlu nesnelerin görsel içerisinden temizlenmesini sağlayan akış diyagramı



Şekil 3.33. Görselde bulunan büyük çatlak olmayan nesnelerin temizlenmesi
A) Çatlak olmayan küçük nesnelerden arındırılmış görsel **B)** Çatlak olmayan büyük nesneler ortadan kaldırılmış görsel

3.3.5. Görsel İçerisinde Çatlak Alanının Tespit Edilmesi

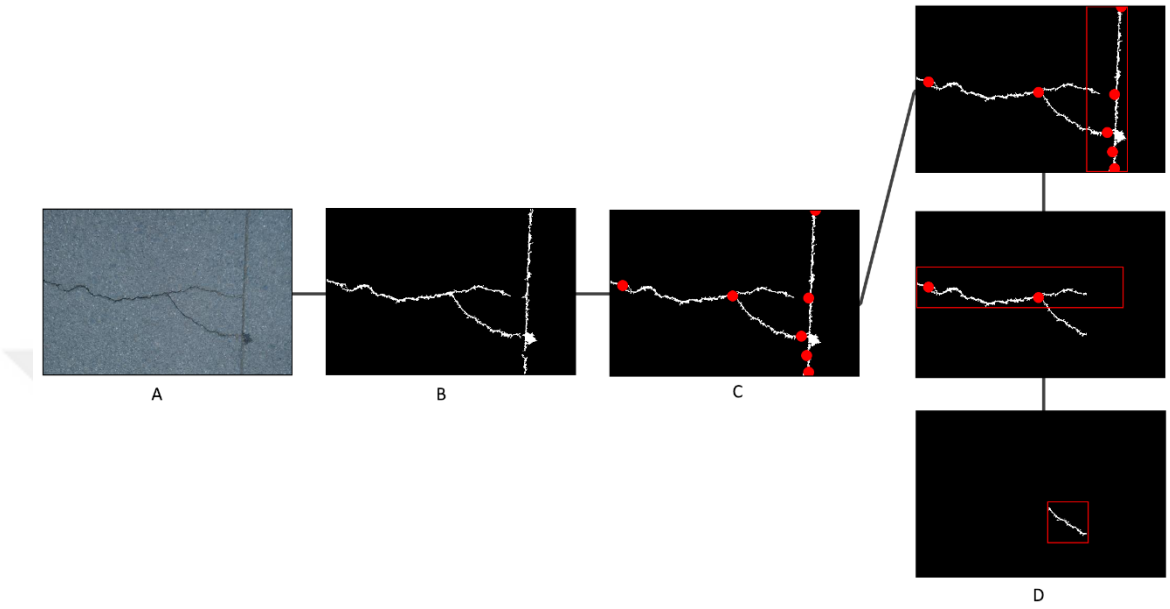
Çatlak tespitinde zamandan yana avantaj sağlamak için GYBE_DSA yaklaşımıyla geliştirilmiş DSA yöntemi kullanılarak elde edilen tespit verileri ile 80 x 80 piksellik bir boyuta sahip bir bölgede çatlak olup olmadığı tespit edilebilmektedir. Ancak mevcut görseller içerisinde bu boyutlardan daha büyük boyutlarda çatlaklar olması muhtemeldir. Bu nedenle ana görsel içinde kabaca belirlenen çatlağın 80 x 80 piksellik alanlara ayrılması gerekmekte ve ardından bu alanlarda DSA verileri kullanılarak çatlak tespiti yapılmalıdır. Uygulanan işlemler Şekil 3.34 ve Şekil 3.35'te gösterilmiştir.

Bahsedildiği üzere çatlak tespiti yapmak için çatlak alanının kabaca belirlenmiş olması gerekmektedir bu nedenle önceden bahsedilen 5 görüntü işleme adımı kullanılarak Şekil 3.34A'da gösterilen görselden çatlak olmayan nesnelerden arındırılmış Şekil 3.34B'de gösterilen görsel elde edilmiştir. Ancak belirlenen alanın nasıl ayrılacağı bilinmemektedir bu nedenle çatlak nesnesinin önemli noktaları iç bükey ve dış bükey mantığına dayanan basit bir kod sistemi kullanılarak Şekil 3.34C'de gösterildiği gibi görsel üzerinde belirlenir. Ardından bu noktaları kapsayan dikey ve yatay alanlar taranır ve alanlar içerisinde en çok bu noktaları kapsayan alan en uygunu olarak belirlenir. Buna takiben belirlenen alan görsel içerisinden çıkarılır ve tekrar alan belirlenir. Bu işlem Şekil 3.34D'de gösterildiği üzere görselde nesne kalmayana kadar devam etmektedir.

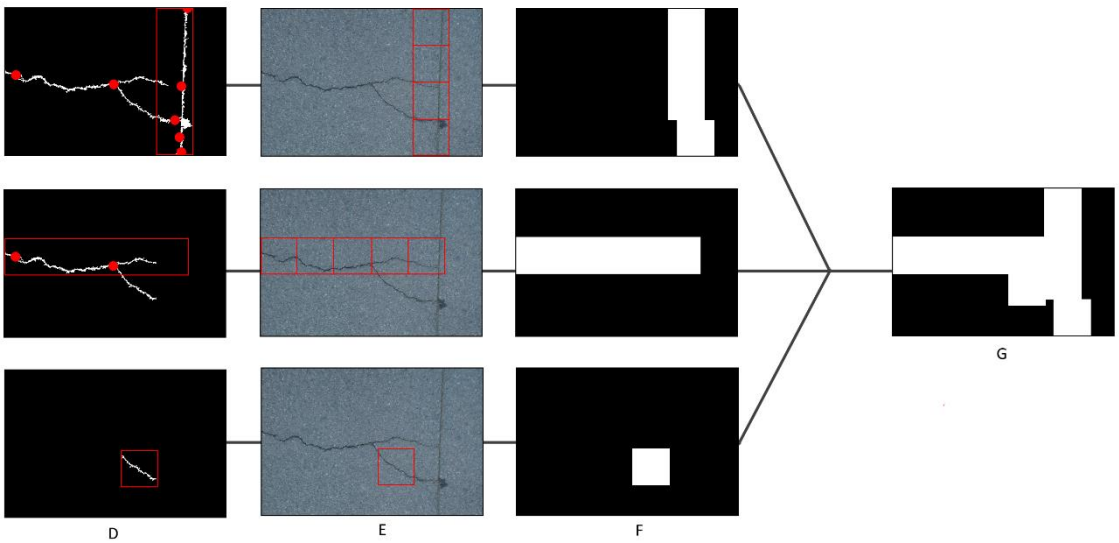
Şekil 3.35D'de gösterilen alanlar belirlendikten sonra alanlar Şekil 3.35E'de gözüken 80 x 80 piksellik bölgelere parçalanır ve bu bölgelerde DSA verileri kullanılarak çatlak tespiti yapılır. Belirlenen bölgelerde çatlak tespit edilmesi durumunda bu bölgeler Şekil 3.35F'de gösterildiği gibi beyaz renk ile belirlenir. Son olarak ta tüm çatlak tespit edilen bölgeler Şekil 3.35G'de gösterildiği gibi birleştirilir ve bir görselin çatlak alanı tespit edilir.

Bu işlemlerin yanında seçilen tarama alanındaki parçalanmış her bir alan içerisinde çatlağın konumu alanın köşelerine denk gelebilmektedir ve bu durum çatlak tespitinin başarısız olmasına neden olmaktadır. Bu sorunu çözmek içinde çatlak tespit edilmeyen her bir 80x80 piksellik tarama alanı 20 piksel boyutunda taranabilecek her kenara doğru

kaydırılır ve çatlak taraması yapılır. Çatlak tespit edilmesi durumunda kaydırma işlemi durdurulur ve taranan alan beyaz renk ile belirlenir. Bu sayede daha iyi konumlandırılmış bir çatlak bölgesi elde edilmektedir.



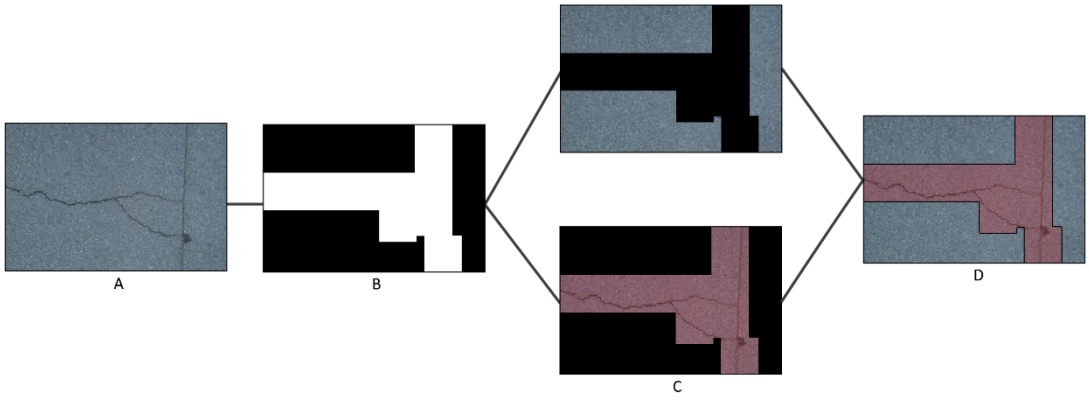
Şekil 3.34. Çatlak tespiti yapılacak görselde çatlak bölgelerin belirlenmesi
A) Orijinal görsel **B)** Çatlak olmayan nesnelerden arındırılmış görsel **C)** Önemli noktaları tespit edilmiş görsel **D)** Çatlak nesneni belirli alanlara parçalanmış görseller



Şekil 3.35. Çatlak bölgeleri belirlenmiş alanlarda çatlak tespitinin yapılması
D) Çatlak nesneni belirli alanlara parçalanmış görseller **E)** Tarama alanı belirlenmiş görseller **F)** Çatlak tespiti yapılan alanları beyaz renk ile belirtilmiş görseller **G)** Çatlak tespit edilen tüm bölgelerin birleştirilmesi ile çatlak bölgesi oluşturulmuş görsel

3.3.5.1. Tespit Edilen Çatlak Bölgesini Ana Resimde Gösterilmesi

Tespit edilen çatlak alanı siyah-beyaz görsel üzerinde gösterilse de ana resimde de bu alanın gösterilmesi gerekmektedir. Bu amaçla Şekil 3.36A'da gösterilen görsele çatlak tespit yöntemi uygulanarak Şekil 3.36B'de gözüken çatlak alanı tespit edilmiş görsele OpenCV kütüphanesinde yer alan "Canny" yöntemi uygulayarak çatlak bölgesinin kenarları tespit edilmektedir. Bu işlemin ardından Şekil 3.36C'de gösterilen iki farklı görsel elde edilmiştir. İlk başta ana görsel içinden çatlak tespiti yapılan bölgeye kırmızı bir filtre uygulanmakta ve ardından filtrelenmiş ve orijinal resme çatlak bölgesine göre maskeleme uygulanmaktadır. Son olarak kırmızı renk ile filtrelenmiş çatlak bölgesi, çatlak bölgesi silinen ana görsel ve çatlak bölgesinin kenarları birleştirilir. Tüm bu işlemlerin sayesinde çatlak tespiti yapılan görsel üzerinde çatlak alanı Şekil 3.36D'de gösterildiği gibi açıkça gösterilmektedir. Önerilen görüntü işleme yöntemlerin PHYTON yazılım kodları EK-3'te sunulmaktadır.



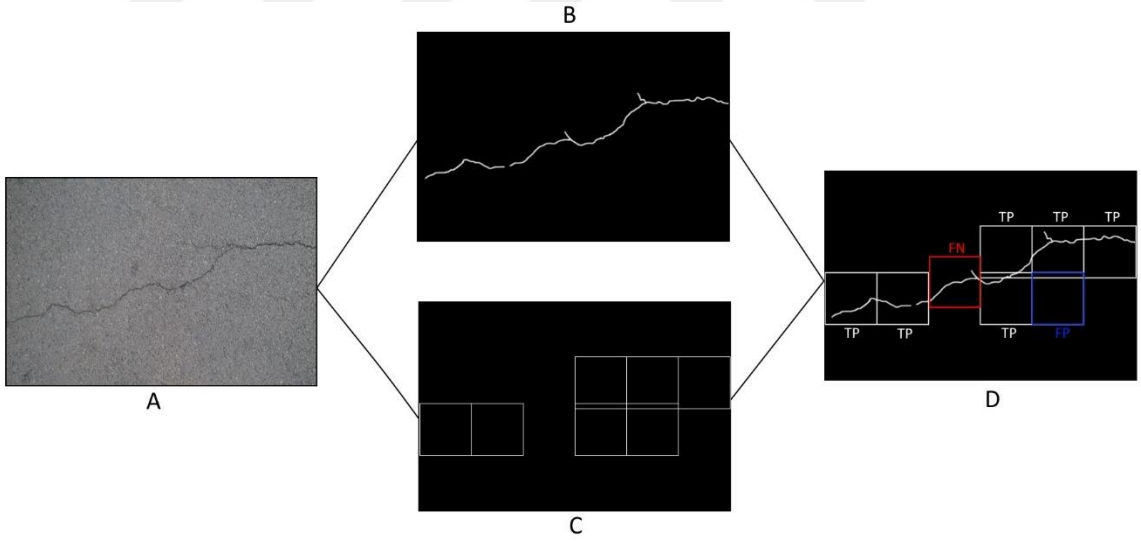
Şekil 3.36. Çatlak bölgesinin orijinal resim üzerinde gösterilmesi
A) Orijinal görsel **B)** Çatlak bölgesi belirlenmiş görsel **C)** Orijinal ve filtrelenmiş resme çatlak bölgesi kullanılarak maskeleme yapılmış görsel **D)** Orijinal görsel üzerinde kırmızı bir alan ile çatlak bölgesi gösterilmiş görsel

3.3.5.2. Önerilen Yöntemin Başarısının Değerlendirilmesi

Geri Yayımlı Birlikte Evrim Yöntemiyle İyileştirilmiş DSA tekniği kullanılarak elde edilen parametreler ile görüntü işleme metotları birlikte uygulanarak çatlak tespiti yapılmıştır. Ancak önerilen yöntemin ne ölçüde başarılı olduğunu göstermek için

başarısının değerlendirilmesi gerekmektedir. Bu amaçla CFD veri seti içerisindeki 80 farklı görselde içerisinde yer alan çatlaklar siyah-beyaz bir görsel üzerinde manuel olarak belirlenmiş ve kaydedilmiştir. Ardından Şekil 3.37A'da gösterilen orijinal görsellerden yararlanarak Şekil 3.37B'de örnek gösterildiği gibi manuel olarak elde edilen görseller ile Şekil 3.37C'de gösterildiği üzere önerilen yöntem kullanılarak çatlak alanları belirlenmiş görseller karşılaştırılmıştır.

Karşılaştırma aşamasında önceki bölümlerde verilen Kesinlik, Doğruluk ve F1-Score başarı değerlendirme yöntemlerinin TP, FP ve FN terimleri Şekil 3.37D'de gösterildiği gibi temsil edilmiştir. Burada TP çatlak tespiti yapılan bölgede çatlak olduğu tahmin edilen ve manuel olarak belirlenmiş görsele göre çatlak bulunduran, FP çatlak tespiti yapılan bölgede çatlak olduğu tahmin edilen ama manuel olarak belirlenmiş görsele göre çatlak bulundurmayan, FN ise çatlak tespiti yapılan bölgede çatlak olmadığı tahmin edilen ve manuel olarak belirlenen görsele göre çatlak bulundurmayan tüm değerlerin toplamıdır.



Şekil 3.37. Çatlak tespiti için olması gereken ve önerilen yöntem ile bulunan çatlak alanları
A) Orijinal görsel **B)** Manuel olarak çatlakları belirlenmiş görsel **C)** Önerilen yöntem kullanılarak çatlak alanı belirlenmiş görsel **D)** TP, FP ve FN değerleri belirlenmiş görsel

4. BULGULAR ve TARTIŞMA

4.1. GYBE_DSA ile Optimum Öğrenme Performansı

Belirlenen eşik değerine göre güncellenmiş veri setine GYBE_DSA yöntemi uygulanması sırasında çeşitli ağ yapıları kullanılmıştır. Ağ yapıları belirlenirken öğrenme durumu ve işlem yapılan sistemin donanımı dikkate alınmıştır. Bu nedenle gizli katmanda öğrenme gerçekleştirilen en az nöron sayısına sahip 6400-100-2 ağ yapısı ile başlanmış ve tek bir gizli katmana sahip her bir ağ yapısı için sistemin öğrenmeyi donanımsal olarak sağlayana kadar 100 tane nöron eklenmiştir. Ayrıca iki gizli katmana sahip ağ yapıları içinde aynı durumlar dikkate alınarak nöron artırımı yapılarak öğrenim gerçekleştirilmiştir. Böylece Çizelge 4.1'de gösterildiği üzere farklı ağ yapıları için farklı başarı durumları elde edilmiştir. Ardından bu başarı durumlarına bakılarak en iyi performansı gösteren ağ yapısı seçilmiş ve en başarılı ağ yapısına ait çıktı verileri daha sonra kullanılmak amacıyla kaydedilmiştir.

Çizelge 4.1. Çeşitli ağ yapılarına göre öğrenme başarısının 3 farklı yöntem ile değerlendirilmesi

Ağ Yapısı	F1-Score	Kesinlik	Duyarlılık
6400-100-2	86,88%	86,79%	86,97%
6400-200-2	**87,38%**	**87,32%**	**87,43%**
6400-300-2	86,61%	86,10%	87,13%
6400-400-2	86,49%	86,24%	86,73%
6400-500-2	86,61%	86,63%	86,60%
6400-50-50-2	87,29%	87,27%	87,30%
6400-100-100-2	87,26%	87,29%	87,23%
6400-200-200-2	86,53%	86,56%	86,50%
6400-300-300-2	86,80%	86,80%	86,80%

Burada koyu renkli olarak belirtilen ve en yüksek başarı değerlerine sahip olan ağ yapısının 6400 değeri giriş katmanındaki nöron değerini temsil ederken son katmandaki 2 değeri çıktı katmanındaki nöronların sayısını belirtmektedir. Arada bulunan sayısal

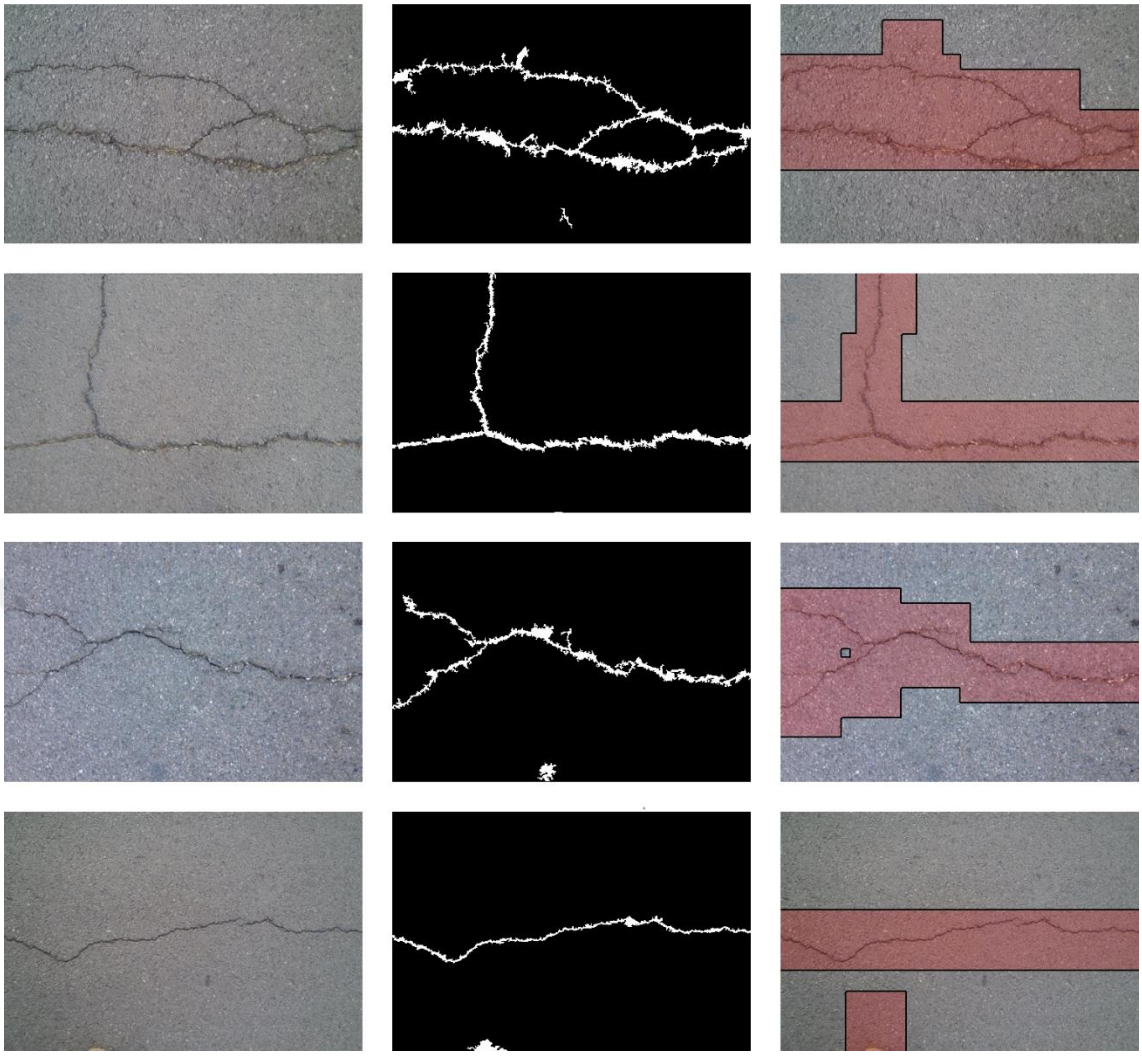
değer ise gizli katman veya gizli katmanların içerisinde yer alan nöronların sayısını belirtmektedir.

Çizelgeye bakıldığında daha güvenilir olarak dikkate alınan F1-Score başarı değerlendirme yönteminde %87,38 ile en yüksek başarıya sahip ağ yapısı 3 katmanlıdır ve gizli katmanında 200 tane nöron bulunmaktadır. Elde edilen başarı oranlarına bakıldığında ise ağ yapıları arasında bir korelasyon olmadığı görülmüştür. Sonuç olarak çalışmada kullanılan veriler için katman veya nöron sayısının artmasının başarıya doğrudan bir katkı sağlamadığı saptanmıştır.

4.2. Önerilen Yöntem ile Çatlak Tespitinin Yapılması

Farklı ağ yapılarına göre başarı değerlendirilmesi yapılmış ve 6400-200-2 ağ yapısına ait ağırlık parametrelerinin daha başarılı olduğu görülmüştür. Belirlenen ağırlık değerleri ile görüntü işleme teknikleri CFD veri setinde bulunan 80 görsel üzerinde test edilmiştir ve Şekil 4.1'de önerilen yöntem ile çatlak alanları tespit edilen bazı görseller gösterilmiştir.

Şekil 4.1'e bakıldığında bazı çatlak bölgelerinin tespit edilmesinde küçük çaplı hataların olduğu gözükmektedir ama çoğunlukla doğru bölgeler taranmıştır. Diğer taraftan önerilen yönteminin Doğruluk, Kesinlik ve F1-Score yöntemleri açısından başarısı CFD veri seti kullanarak farklı yöntemler öneren çeşitli çalışmaların başarısı ile birlikte Çizelge 4.2'de verilmektedir.



Şekil 4.1. Çatlak tespiti yapılan görsellerinin sırasıyla orijinal, çatlak olmayan nesnelerden arındırılmış ve çatlak tespit edilen bölgeleri belirtilmiş halleri

Çizelge 4.2. CFD veri seti kullanarak çatlak tespiti yapan yöntemler ve tespit başarıları

Yöntem	Kesinlik	Duyarlılık	F1-Score
CrackTree	%73,22	%76,45	%70,80
CrackForest	%82,28	%89,44	%85,71
FFA	%78,56	%68,43	%73,15
CrackNet-V	%92,58	%86,03	%89,18
ConnCrack	%96,79	%87,75	%91,96
Önerilen Yöntem	%92,74	%88,92	%89,61

Burada CrackTree (Q. Zou ve diğerleri, 2012) minimum kapsamlı ağaç yapısına dayanan bir yöntem iken CrackForest (Shi ve diğerleri, 2016) rastgele yapılandırılmış ağaç

modelini temel almaktadır. FFA (Nguyen, Begot, Duculty ve Avila, 2011) ise çatlak tespiti için her türlü serbest biçimli ve yol boyunca hesaplanan özellikleri kullanmaktadır. CrackNet-V (Fei ve diğerleri, 2020) yönteminde girdilerin boyutunu yedeklemek ve çatlakları tespit etmek için havuzlama katmanı kullanmayan bir yapı önerilmektedir. ConnCrack (Mei ve Gül, 2020) metodu ise koşullu Wasserstein Çekişmeli Üretici Ağ ile bağlantı haritaları tekniğini birleştirmektedir.

Çizelge 4.2'de görüldüğü üzere önerilen yöntem CrackTree, CrackForest ve FFA yöntemlerinden daha iyi başarıya sahipken CrackNet-V yöntemiyle benzer derecede başarıya ulaşabilmiştir. ConnCrack yöntemi ise Kesinlik ve F1-Score değerleri açısından daha başarılı bir yöntem olduğu açıkça görülmektedir.

Bu yöntemler sadece çatlak tespitinde başarıya odaklanmış ve çatlak alanından ziyade doğrudan çatlağı tespit etmeye çalışan dolayısıyla farklı başarı değerlendirme yolları izleyen metotlardır ve önerdikleri yöntemlerin çatlak tespit etme süreleri hakkında bilgi verilmemiştir. Ayrıca önerdiğimiz yöntem tespit başarısının yanında gerçek zamanlı tespit etmeyi amaçlayan bir metot olduğu için önerilen yöntemin başarısının çizelgede verilen yöntemlerle doğrudan karşılaştırılması çok uygun bir yaklaşım olmayacağı ancak başarı durumu hakkında genel bir bilgi verebileceği düşünülmüştür.

Ek olarak genel başarıya katkıda bulunmak amacıyla gerçek zamanlı çatlak tespiti yapmaya amaçlayan yöntemler ile de karşılaştırma yapılmıştır. Bu amacı yerine getirmek için öncelikle başarının tespit edildiği veri seti üzerinde Çizelge 4.3'te verilen iki farklı cihaz kullanılarak minimum, ortalama ve maksimum saniyelik görüntü sayısı Çizelge 4.4'te gösterildiği üzere hesaplanmıştır.

Çizelge 4.3. Çatlak tespiti yapılan bilgisayarların teknik özellikleri

	RAM	**CPU**	**GPU**
1.Cihaz	8 GB	Intel i7-7500U @2.70GHz, 2 Çekirdek 4 Mantıksal İşlemci	AMD Radeon R7 M440
2.Cihaz	16 GB	Intel i7-10750H @2.60GHz, 6 Çekirdek 12 Mantıksal İşlemci	NVIDIA GeForce GTX 1650

Çizelge 4.4. Çatlak tespiti yapan cihazların FBS değerleri

	1.Cihaz	**2.Cihaz**
Minimum (FBS)	25,0	47,7
Ortalama (FBS)	12,7	19,3
Maksimum (FBS)	3,1	5,3

Yukarıda tabloda görüldüğü üzere 1.Cihaz'dan daha iyi teknik özelliklere sahip olan 2.Cihaz daha yüksek FBS değerlerine erişebilmiş ve neredeyse 48 FBS ile iki katı daha iyi başarı göstermiştir. Ancak bu bilgi tek başına genel başarıya katkıda bulanamamaktadır bu nedenle önerilen yöntemin F1-Score başarı değerleri ve maksimum FBS değerleri Çizelge 4.5'te gösterilen bazı yöntemlerin sonuçlarıyla karşılaştırılmıştır.

Çizelge 4.5. Çatlak tespiti açısından bazı yöntemlerin başarısı ve FBS değerleri

Yöntem	**F1-Score**	**FBS**
EfficientDetDD	%56,00	180
DenseASPP	%81,71	14,2
SegNet	%81,91	15,5
DLCS	%82,70	46,4
MFLCD	%98,22	96,6
Önerilen Yöntem	**%88,97**	**47,7**

Burada EfficientDetDD (Naddaf-Sh ve diğerleri, 2020) yol kusurlarını gerçek zamanlı olarak tespit etmek amacıyla ölçeklenebilir ve verimli modeller eğitmek için derin öğrenme tabanlı bir şemayken DenseASPP (Yang, Yu, Zhang, Li ve Yang, 2018) yoğun

bir şekilde çok ölçekli özellik üreten bir dizi atröz evrişimli katmanları birbirine bağlayan bir yöntemdir. SegNet (Badrinarayanan, Kendall ve Cipolla, 2017) ise semantik piksel bazında segmentasyon için derin tamamen evrişimli bir sinir ağı mimarisi kullanmaktadır. Son olarak, DLCS (W. Wang ve Su, 2021) ikili segmentasyon ağına dayalı yüzeysel bir çatlak segmentasyon modeliyken MFLCD (Ma ve diğerleri, 2021) birden çok özellik katmanına sahip evrişimli sinir ağına dayalı bir çatlak tespit sistemidir.

Yukarıda verilen çizelgeye bakıldığında önerdiğimiz yöntem DenseASPP, SegNet ve DLCS yöntemlerinden başarılı olsa da EfficientDetDD yönteminden sadece F1-Score açısından başarılıdır. MFLCD yönteminin ise açık bir şekilde en iyi yöntem olduğu görülmektedir. Ancak çalışmalarda kullanılan veri setleri ve tespit yapılan cihazın özelliklerinin FBS'i önemli ölçüde etkileyebileceği Çizelge 4. 4'te açık bir şekilde gösterilmiştir ve MFLCD yönteminde kullanılan teknik cihaz (Intel i7-7800x @3.50GHz, 6 Çekirdek 12 Mantıksal İşlemci CPU, 64 GB RAM, NVIDIA Titan V GPU) bu çalışmada kullanılan cihazlardan belirgin bir şekilde daha iyi bir donanıma sahip olması diğer yöntemlerle önerdiğimiz yöntemi doğrudan karşılaştırmanın gerçekçi bir değerlendirme olmayacağını göstermektedir. Ancak yukarıda belirtildiği üzere yapılan karşılaştırmalar yaptığımız çalışmanın başarı durumu hakkında genel bir bilgi verebilmektedir.

5. SONUÇ

Bu çalışmada GYBE_DSA yöntemi ile görüntü işleme teknikleri kullanılarak çatlak tespiti yeni bir yöntem önerilmiştir. Bu kapsamda çatlak görsel verileri sistem içerisinde tanımlanarak çatlak tespitinin yapılması öğretilmiş ve öğrenen model kullanılarak yeni görsel veriler üzerinde çatlak tespiti yapılabilmiştir. Ancak model ile sadece 80 x 80 piksel boyutlarına sahip görsellerde çatlak olup olmadığı belirlenmektedir. Bunun yanında çatlak tespiti yapılmak istenen görseller çok daha büyük boyuttalardır ve görseller içerisinde çatlağın konumu tam olarak bilinmemektedir. Bu amaçla ilk başta çatlak tespiti yapılmak istenen görsele Uyarlanabilir Eşikleme Yöntemi uygulanmaktadır. Ardından görsel içerisinde yer alan çok küçük boyutlarda çatlak olmayan nesneler temizlenmekte ve nesnelerin içerisinde yer alan çok küçük boşluklar doldurulmaktadır. Bu işlemin ardından küçük boyutta bulunan nesneler, veri setindeki görsellerde yer alan nesnelerin geometrik özelliklerinden yararlanarak görsel içerisinden çıkarılmaktadır. Bu işleme takiben birbirine oldukça yakın olan çatlak nesneleri birleştirilmekte ve büyük boyutlu çatlak içermeyen nesneler çeşitli geometrik özellik değerleri kullanılarak silinmektedir. Son olarak da görsel içerisinde yer alan bölge parçalanarak taranacak alanlar belirlenmekte ve belirlenen bölgelerde GYBE_DSA tekniğiyle öğrenilen parametreler kullanılarak çatlak tespiti yapılmaktadır.

Çalışmada GYBE_DSA yöntemi içerisinde 6400-200-2 ağ yapısına sahip modelin en yüksek başarı değerlerine sahip olduğu gözlenmiştir. Belirlenen en başarılı model ile CFD veri seti üzerinde maksimum saniyede 48 görsel üzerinde çatlak tespiti yapılabilmektedir. Yöntemin başarı yüzdeleri Kesinlik için %92,74, Duyarlılık için %88,92 ve F1-Score için %89,61 çıkmıştır. Bu yöntem küçük bir alanda çatlak tespit etme özelliği ile farklı boyutlara sahip çatlak görselleri üzerinde kullanılabilmektedir. Çatlak tespiti sürecinde harcanan süre maliyeti kameraların genel olarak 30 ile 60 FBS arasında teknik özelliklere sahip olması nedeniyle gerçek zamanlı tespit için yeterli görünmekte ve tespit yapacak bilgisayar veya cihaz daha iyi teknik özelliklere sahip olursa tespit süresinin daha da azalacağı ve FBS değerinin artacağı düşünülmektedir.

Yöntemin yaygın etkisini arttırmak için GPS ve kamera sistemi ile şehrin her bölgesine seyahat eden araçlarda kullanılmalıdır. Bu şekilde hem yeni görsel veri setleri elde edilebilir hem de yol üzerinde tespit edilen her bir çatlağın konumu belirlenerek bölgesel olarak çatlak durumu analiz edilebilir. Sonuç olarak yol çalışması yapacak olan kamu kuruluşları ve özel firmalar öncelikli bölgeyi doğrudan belirleyerek çalışmalarını bu bilgiye göre planlayabilirler.

KAYNAKLAR

Amhaz, R., Chambon, S., Idier, J. ve Baltazart, V. (2016). Automatic Crack Detection on Two-Dimensional Pavement Images: An Algorithm Based on Minimal Path Selection. *IEEE Transactions on Intelligent Transportation Systems*, *17*(10), 2718–2729. doi:10.1109/TITS.2015.2477675

Anonim. (y.y.). Dijital Görüntü İşleme. https://en.wikipedia.org/wiki/Digital_image_processing adresinden erişildi.

Arya, D., Maeda, H., Ghosh, S. K., Toshniwal, D. ve Sekimoto, Y. (2021). RDD2020: An annotated image dataset for automatic road damage detection using deep learning. *Data in Brief*, *36*, 107133. doi:10.1016/j.dib.2021.107133

Badrinarayanan, V., Kendall, A. ve Cipolla, R. (2017). SegNet: A Deep Convolutional Encoder-Decoder Architecture for Image Segmentation. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, *39*(12), 2481–2495. doi:10.1109/TPAMI.2016.2644615

Balaji, A. J., Thiru Balaji, G., DInesh, M. S., Nair, B. B. ve Harish Ram, D. S. (2018). A Machine Learning Based Approach to Crack Detection in Asphalt Pavements. *INDICON 2018 - 15th IEEE India Council International Conference*, 9–12. doi:10.1109/INDICON45594.2018.8987039

Cao, W., Zou, Y., Luo, M., Zhang, P., Wang, W. ve Huang, W. (2019). Deep Discriminant Learning-based Asphalt Road Cracks Detection via Wireless Camera Network. *2019 Computing, Communications and IoT Applications, ComComAp 2019*, 53–58. doi:10.1109/ComComAp46287.2019.9018831

Eisenbach, M., Stricker, R., Seichter, D., Amende, K., Debes, K., Sesselmann, M., … Gross, H. M. (2017). How to get pavement distress detection ready for deep learning? A systematic approach. *Proceedings of the International Joint Conference on Neural Networks, 2017-May*, 2039–2047. doi:10.1109/IJCNN.2017.7966101

Fei, Y., Wang, K. C. P., Zhang, A., Chen, C., Li, J. Q., Liu, Y., … Li, B. (2020). Pixel-Level Cracking Detection on 3D Asphalt Pavement Images Through Deep-Learning- Based CrackNet-V. *IEEE Transactions on Intelligent Transportation Systems*, *21*(1), 273–284. doi:10.1109/TITS.2019.2891167

Feng, J., He, X., Teng, Q., Ren, C., Chen, H. ve Li, Y. (2019). Reconstruction of porous media from extremely limited information using conditional generative adversarial networks. *Physical Review E*, *100*(3), 033308. doi:10.1103/PhysRevE.100.033308

Gavilán, M., Balcones, D., Marcos, O., Llorca, D. F., Sotelo, M. A., Parra, I., … Amírola, A. (2011). Adaptive Road Crack Detection System by Pavement Classification. *Sensors*, *11*(10), 9628–9657. doi:10.3390/s111009628

Gong, M., Liu, J., Qin, A. K., Zhao, K. ve Tan, K. C. (2021). Evolving Deep Neural Networks via Cooperative Coevolution with Backpropagation. *IEEE Transactions on Neural Networks and Learning Systems*, *32*(1), 420–434. doi:10.1109/TNNLS.2020.2978857

Goodfellow, I., Pouget-Abadie, J., Mirza, M., Xu, B., Warde-Farley, D., Ozair, S., … Bengio, Y. (2014). Generative adversarial networks. *Communications of the ACM*, *63*(11), 139–144. doi:10.1145/3422622

Han, Z., Chen, H., Liu, Y., Li, Y., Du, Y. ve Zhang, H. (2021). Vision-Based Crack Detection of Asphalt Pavement Using Deep Convolutional Neural Network. *Iranian Journal of Science and Technology - Transactions of Civil Engineering*, *45*(3),

2047–2055. doi:10.1007/s40996-021-00668-x

Hassan, S. A., Han, S. H. ve Shin, S. Y. (2020). Real-time Road Cracks Detection based on Improved Deep Convolutional Neural Network. *Canadian Conference on Electrical and Computer Engineering*, *2020-Augus*, 5–8. doi:10.1109/CCECE47787.2020.9255771

Hinton, G. E., Osindero, S. ve Teh, Y.-W. (2006). A Fast Learning Algorithm for Deep Belief Nets. *Neural Computation*, *18*(7), 1527–1554. doi:10.1162/neco.2006.18.7.1527

Hoang, N.-D. (2018). An Artificial Intelligence Method for Asphalt Pavement Pothole Detection Using Least Squares Support Vector Machine and Neural Network with Steerable Filter-Based Feature Extraction. *Advances in Civil Engineering*, *2018*, 1–12. doi:10.1155/2018/7419058

Hornik, K., Stinchcombe, M. ve White, H. (1989). Multilayer feedforward networks are universal approximators. *Neural Networks*, *2*(5), 359–366. doi:10.1016/0893-6080(89)90020-8

HU, Y., ZHAO, C. ve WANG, H. (2010). Automatic Pavement Crack Detection Using Texture and Shape Descriptors. *IETE Technical Review*, *27*(5), 398. doi:10.4103/0256-4602.62225

JaChing Chou, O'Neill, W. A. ve Cheng, H. D. (1994). Pavement distress classification using neural networks. *Proceedings of IEEE International Conference on Systems, Man and Cybernetics* içinde (C. 1, ss. 397–401). IEEE. doi:10.1109/ICSMC.1994.399871

Jayanth Balaji, A., Thiru Balaji, G., Dinesh, M. S., Binoy, N. ve Harish Ram, D. S. (2019). Asphalt Crack Dataset. *Mendeley Data*. doi:10.17632/xnzhj3x8v4.2

Karjala, T. W., Himmelblau, D. M. ve Miikkulainen, R. (1992). Data rectification using recurrent (Elman) neural networks. *[Proceedings 1992] IJCNN International Joint Conference on Neural Networks* içinde (C. 2, ss. 901–906). IEEE. doi:10.1109/IJCNN.1992.226873

Kaseko, M. S. ve Ritchie, S. G. (1993). A neural network-based methodology for pavement crack detection and classification. *Transportation Research Part C: Emerging Technologies*, *1*(4), 275–291. doi:10.1016/0968-090X(93)90002-W

LeCun, Y., Bottou, L., Bengio, Y. ve Haffner, P. (1998). Gradient-based learning applied to document recognition. *Proceedings of the IEEE*, *86*(11), 2278–2323. doi:10.1109/5.726791

Lee, B. J. ve Lee, H. "David". (2004). Position-Invariant Neural Network for Digital Pavement Crack Analysis. *Computer-Aided Civil and Infrastructure Engineering*, *19*(2), 105–118. doi:10.1111/j.1467-8667.2004.00341.x

Li, N., Hou, X., Yang, X. ve Dong, Y. (2009). Automation Recognition of Pavement Surface Distress Based on Support Vector Machine. *2009 Second International Conference on Intelligent Networks and Intelligent Systems* içinde (ss. 346–349). IEEE. doi:10.1109/ICINIS.2009.95

Lyu, P. H., Wang, J. ve Wei, R. Y. (2019). Pavement crack image detection based on deep learning. *PervasiveHealth: Pervasive Computing Technologies for Healthcare*, 6–10. doi:10.1145/3342999.3343003

Ma, D., Fang, H., Wang, N., Xue, B., Dong, J. ve Wang, F. (2021). A real-time crack detection algorithm for pavement based on CNN with multiple feature layers. *Road Materials and Pavement Design*, (May). doi:10.1080/14680629.2021.1925578

Mandal, V., Uong, L. ve Adu-Gyamfi, Y. (2019). Automated Road Crack Detection

Using Deep Convolutional Neural Networks. *Proceedings - 2018 IEEE International Conference on Big Data, Big Data 2018*, 5212–5215. doi:10.1109/BigData.2018.8622327

Mathavan, S., Rahman, M. ve Kamal, K. (2015). Use of a Self-Organizing Map for Crack Detection in Highly Textured Pavement Images. *Journal of Infrastructure Systems*, *21*(3). doi:10.1061/(ASCE)IS.1943-555X.0000237

McCulloch, W. S. ve Pitts, W. (1943). A logical calculus of the ideas immanent in nervous activity. *The Bulletin of Mathematical Biophysics*, *5*(4), 115–133. doi:10.1007/BF02478259

Mei, Q. ve Gül, M. (2020). A cost effective solution for pavement crack inspection using cameras and deep neural networks. *Construction and Building Materials*, *256*, 119397. doi:10.1016/j.conbuildmat.2020.119397

Mei, Q., Gül, M. ve Azim, M. R. (2020). Densely connected deep neural network considering connectivity of pixels for automatic crack detection. *Automation in Construction*, *110*(November 2019), 103018. doi:10.1016/j.autcon.2019.103018

Mei, Q., Gül, M. ve Shirzad-Ghaleroudkhani, N. (2020). Towards smart cities: crowdsensing-based monitoring of transportation infrastructure using in-traffic vehicles. *Journal of Civil Structural Health Monitoring*, *10*(4), 653–665. doi:10.1007/s13349-020-00411-6

Naddaf-Sh, S., Naddaf-Sh, M. M., Kashani, A. R. ve Zargarzadeh, H. (2020). An Efficient and Scalable Deep Learning Approach for Road Damage Detection. *Proceedings - 2020 IEEE International Conference on Big Data, Big Data 2020*, 5602–5608. doi:10.1109/BigData50022.2020.9377751

Nguyen, T. S., Begot, S., Duculty, F. ve Avila, M. (2011). Free-form anisotropy: A new method for crack detection on pavement surface images. *2011 18th IEEE International Conference on Image Processing* içinde (ss. 1069–1072). IEEE. doi:10.1109/ICIP.2011.6115610

O'Byrne, M., Schoefs, F., Ghosh, B. ve Pakrashi, V. (2013). Texture Analysis Based Damage Detection of Ageing Infrastructural Elements. *Computer-Aided Civil and Infrastructure Engineering*, *28*(3), 162–177. doi:10.1111/j.1467-8667.2012.00790.x

Potter, M. A. ve De Jong, K. A. (1994). A cooperative coevolutionary approach to function optimization. *Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics)*, *866 LNCS*, 249–257. doi:10.1007/3-540-58484-6_269

Qin, A. K., Huang, V. L. ve Suganthan, P. N. (2009). Differential evolution algorithm with strategy adaptation for global numerical optimization. *IEEE Transactions on Evolutionary Computation*, *13*(2), 398–417. doi:10.1109/TEVC.2008.927706

Redmon, J., Divvala, S., Girshick, R. ve Farhadi, A. (2016). You only look once: Unified, real-time object detection. *Proceedings of the IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, *2016-Decem*, 779–788. doi:10.1109/CVPR.2016.91

Ronneberger, O., Fischer, P. ve Brox, T. (2015). U-Net : Convolutional Networks for Biomedical Image Segmentation. *LECTURE NOTES IN ARTIFICIAL INTELLIGENCE*, 234–241. doi:10.1007/978-3-319-24574-4

Rumelhart, D. E., Hinton, G. E. ve Williams, R. J. (1986). Learning Internal Representations by Error Propagation. *Readings in Cognitive Science: A Perspective from Psychology and Artificial Intelligence*, *1*(V), 399–421. doi:10.1016/B978-1-4832-1446-7.50035-2

Shi, Y., Cui, L., Qi, Z., Meng, F. ve Chen, Z. (2016). Automatic road crack detection using random structured forests. *IEEE Transactions on Intelligent Transportation Systems*, *17*(12), 3434–3445. doi:10.1109/TITS.2016.2552248

Simonyan, K. ve Zisserman, A. (2014). Very Deep Convolutional Networks for Large-Scale Image Recognition.

Sizyakin, R., Voronin, V. V., Gapon, N. ve Pižurica, A. (2020). A deep learning approach to crack detection on road surfaces, 20. doi:10.1117/12.2574131

Solomon, C. ve Breckon, T. (2010). *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. Wiley-Blackwell. doi:10.1002/9780470689776

Wang, S., Qiu, S., Wang, W., Xiao, D. ve Wang, K. C. P. (2017). Cracking Classification Using Minimum Rectangular Cover–Based Support Vector Machine. *Journal of Computing in Civil Engineering*, *31*(5). doi:10.1061/(ASCE)CP.1943-5487.0000672

Wang, W. ve Su, C. (2021). Deep Learning-Based Real-Time Crack Segmentation for Pavement Images. *KSCE Journal of Civil Engineering*, *25*(12), 4495–4506. doi:10.1007/s12205-021-0474-2

Xin Yao. (1999). Evolving artificial neural networks. *Proceedings of the IEEE*, *87*(9), 1423–1447. doi:10.1109/5.784219

Yang, M., Yu, K., Zhang, C., Li, Z. ve Yang, K. (2018). DenseASPP for Semantic Segmentation in Street Scenes. *2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition* içinde (ss. 3684–3692). IEEE. doi:10.1109/CVPR.2018.00388

Zhang, A., Wang, K. C. P., Fei, Y., Liu, Y., Chen, C., Yang, G., … Qiu, S. (2018). Automated Pixel-Level Pavement Crack Detection on 3D Asphalt Surfaces with a Recurrent Neural Network. *Computer-Aided Civil and Infrastructure Engineering*, *34*(3), 213–229. doi:10.1111/mice.12409

Zhang, A., Wang, K. C. P., Li, B., Yang, E., Dai, X., Peng, Y., … Chen, C. (2017). Automated Pixel-Level Pavement Crack Detection on 3D Asphalt Surfaces Using a Deep-Learning Network. *Computer-Aided Civil and Infrastructure Engineering*, *32*(10), 805–819. doi:10.1111/mice.12297

Zhang, L., Yang, F., Daniel Zhang, Y. ve Zhu, Y. J. (2016). Road crack detection using deep convolutional neural network. *Proceedings - International Conference on Image Processing, ICIP*, *2016-Augus*, 3708–3712. doi:10.1109/ICIP.2016.7533052

Zou, Q., Cao, Y., Li, Q., Mao, Q. ve Wang, S. (2012). CrackTree: Automatic crack detection from pavement images. *Pattern Recognition Letters*, *33*(3), 227–238. doi:10.1016/j.patrec.2011.11.004

Zou, Y., Cao, W., Luo, M., Zhang, P., Wang, W. ve Huang, W. (2019). Deep Learning-based Pavement Cracks Detection via Wireless Visible Light Camera-based Network. *2019 Computing, Communications and IoT Applications, ComComAp 2019*, 47–52. doi:10.1109/ComComAp46287.2019.9018814

EKLER

EK 1 GYBE_DSA yönteminin PHYTON yazılım kodları

EK 2 GYBE_DSA yöntemi için Kesinlik, Duyarlılık ve F1-Score başarı değerlendirme yöntemlerinin PHYTON yazılım kodları

EK 3 Önerilen çatlak tespit yönteminin PHYTON yazılım kodları

EK 4 Önerilen çatlak tespit yöntemi için Kesinlik, Duyarlılık ve F1-Score başarı değerlendirme yöntemlerinin PHYTON yazılım kodları

EK 1

```
__main__.py
-----------

import cv2
import numpy as np
import glob
from skimage import io, morphology
from sklearn.model_selection import train_test_split

from functions import *


# 80*80 boyutunda çatlak içeren ve içermeyen görsellerin uzantılarının
içeriye aktarılması

#Çatlak içeren veriler
crack_paths= glob.glob(r"New_Dataset_80_80\Cracks\AllDatasets\*.png")

#Çatlak içermeyen veriler
non_crack_paths=
glob.glob(r"New_Dataset_80_80\NonCracks\AllDatasets\*.jpg")

# Verilerin içeri aktarılıp thresholdimage fonksiyonun uygulanması ve data
olarak tek bir değişkende tutulması
# [0,1] =çatlak değil [1,0]=çatlak

data= []
for path in crack_paths:
    img= cv2.imread(path,0)
    img_cm=thresholdimage(img)
    x= img_cm.reshape(img.shape[0]*img.shape[1],)
    data.append((x))

for path in non_crack_paths:
    img= cv2.imread(path,0)
    img_cm=thresholdimage(img)
    x= img_cm.reshape(img.shape[0]*img.shape[1],)
    data.append((x))

# Datanın çıktı değerlerini düzenleyerek eğitim ve test olarak ana veri
setinin ayrılması

a=np.zeros((len(data),len(data[0])))
for i in range(len(data)):
    a[i,:]=data[i]
data_n=a
a=np.tile([1,0],(7500,1))
b=np.tile([0,1],(7500,1))
Y=np.append(a,b,axis=0)
```

```
X=data_n
train_X, test_X, train_y, test_y = train_test_split(X, Y, test_size=0.20,
random_state=42)

# Oluştulacak sinir ağının özelliklerinin belirlenmesi

train_x = np.double(train_X)
test_x  = np.double(test_X)
train_y = np.double(train_y)
test_y  = np.double(test_y)

batchsize = 20

numinput = train_x.shape[1]
numhid_1 = 100
numoutput = train_y.shape[1]
m = np.size(train_x,0)

sizes = [numinput,numhid_1,numoutput]

numbatches = m/batchsize

numepochs = 50
iteration = 100
Threshold = 0.01
alph = 0.3
m_ratio = 0.3

# Ağırlık ve yanlılık terimlerinin üretilmesi

nn_size                = sizes
nn_number_layer        = np.size(nn_size) - 1

nn_ll                  = 2
nn_l                   = 0.00002
nn_l2                  = 0.000001

nn_W=[]
nn_b=[]

for i in range(1,nn_number_layer+1):

    nn_Wx = (np.random.rand(nn_size[i], nn_size[i - 1]) - 0.5) * 2 * 4 *
np.sqrt(6 / (nn_size[i] + nn_size[i - 1]))
    nn_W.append(nn_Wx)
    nn_bx = (np.random.rand(nn_size[i],1) - 0.5)*2
    nn_b.append(nn_bx)

Cost_all = np.zeros((numepochs,1))
Cost_index = np.zeros((numepochs,1))

iter_index = 1
subtask_number = sum(sizes[1:])
cost_last = 0
```

```
Wgrad= [[[0]],[[0]],[[0]]]
Delta= [[[0]],[[0]]]

# Elde edilen özellikler değerlerinin kullanılması ile öğrenme işleminin
yapılması

for k in range(0,numepochs):
    print(k)

    kk = np.random.permutation(range(m))

    x = train_x[kk[:iteration*batchsize],:]
    y = train_y[kk[:iteration*batchsize],:]

    for j in range(0,100):

        output=[]

        batch_x = x[(j)*batchsize:batchsize+batchsize*(j), :]
        batch_y = y[(j)*batchsize:batchsize+batchsize*(j),:]

        numpatches = np.size(batch_x.T,1)

        input_x = batch_x.T

        for i in range(0,nn_number_layer):
            output1= sigm(np.dot(nn_W[i], input_x) + np.tile(nn_b[i], (1,
numpatches)))
            output.append(output1)
            input_x = output[i]

        Delta[nn_number_layer-1] = (output[nn_number_layer-1] -
batch_y.T)*output[nn_number_layer-1]*(1 - output[nn_number_layer-1])
        for i in range(nn_number_layer-2,-1,-1):
            Delta[i] = np.dot(nn_W[i+1].T,Delta[i+1])*output[i]*(1 -
output[i])

        Wgrad[0] =  nn_l*nn_W[0] + np.dot(Delta[0],batch_x/numpatches)
        for i in range(1,nn_number_layer):
            Wgrad[i] = nn_l*nn_W[i] + np.dot(Delta[i],(output[i-1].T)/
numpatches)

        for i in range(0,nn_number_layer):
            nn_W[i] = nn_W[i] - nn_ll*Wgrad[i]
            nn_b[i] = nn_b[i] - np.expand_dims(nn_ll*np.sum(Delta[i],1)/
numpatches, axis=1)

    cost = cal_cost(x.T, y.T, nn_number_layer, nn_W, nn_b, nn_l2)

    Cost_all[k,0] = cost
```

```
    nn=[]
    if abs(cost - cost_last) < Threshold/np.sqrt(k+1):

        [_, _, subtask_index_need] = cal_maturity(x.T, nn_number_layer,
nn_W, nn_b, sizes, alph, m_ratio)

        nn.extend((nn_number_layer,nn_W,nn_b, nn_l2))
        globals()['nn_cc_' + str(iter_index)] = nn
        iter_index = iter_index + 1
        globals()['nn_cc_' + str(iter_index)] = saCCDE_all(x.T, y.T,
globals()['nn_cc_' + str(iter_index-1)], sizes, subtask_index_need)

        nn = globals()['nn_cc_' + str(iter_index)]

        nn_number_layer= nn[0]
        nn_W= nn[1]
        nn_b= nn[2]
        nn_l2= nn[3]
        Cost_all[k,0] = cal_cost(x.T, y.T, nn_number_layer,nn_W,nn_b,
nn_l2)

    print(cost_last, cost)
    cost_last = cost
```

```
functions.py
-----------

import cv2
import numpy as np
from skimage import morphology

def thresholdimage(img):
    threshold= cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 135, 18)

    if np.count_nonzero(threshold)>6000 or
np.count_nonzero(threshold)<400:
        threshold= cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 135, 5)

    erosion_w= cv2.bitwise_not(threshold)
    arr = erosion_w > 0
    cleaned = morphology.remove_small_objects(arr, min_size=150)
    cleaned1 = morphology.remove_small_holes(cleaned, 150)
    result= 1*cleaned1
    return result


def sigm(P):
    X = 1/(1+np.exp(-P))
    return X


def g_f(T, alph):
    w = np.sum(np.logical_or((T<alph),T>(1-alph)),1)
    return w


def cal_cost(train_x,train_y, nn_number_layer, nn_W, nn_b, nn_l2):
    N = np.size(train_x,1)
    x = train_x

    for i in range(0,nn_number_layer):
        x= sigm(np.dot(nn_W[i],x) + np.tile(nn_b[i], (1, N)))

    total_sum = 0
    for i in range(0,nn_number_layer):
        total_sum = total_sum + sum(sum(nn_W[i]*nn_W[i]))

    cost = sum(sum((x - train_y)**2))/N/2 + nn_l2*total_sum
    return cost


def get_b(nn,sizes,index):
    layer_number = 0
    nn_number_layer= nn[0]
    nn_W= nn[1]
    nn_b= nn[2]
```

```
    for i in range(0,nn_number_layer):
       if (index < sizes[i+1]) or (index == sizes[i+1]):
            w = nn_W[layer_number][index,:]
            b = nn_b[layer_number][index]
       else:
           index = index - sizes[i+1]

    w_b = np.zeros((np.size(w, axis=1)+1,1))
    w_b[0:np.size(w, axis=1)] = w.T
    w_b[-1] = b
    return w_b


def initial_pop(w_b,N):
    d_m = max(w_b.shape)
    pop_1 = np.zeros((d_m,N))
    w_b1= w_b.reshape((np.size(w_b,axis=0),))
    pop_1[:,0] = w_b1

    for i in range(0,N-1):
        a = (np.random.rand(d_m,1)-0.5)+1
        w_b_new = a*w_b
        w_b_new1= w_b.reshape((np.size(w_b_new,axis=0),))
        pop_1[:,i+1] = w_b_new1

    return pop_1


def cal_maturity(x, nn_number_layer, nn_W, nn_b, sizes, alph, m_ratio):
    number_hidden = sum(sizes[1:])
    split_number = 2

    b_size = int(x.shape[1]/split_number)
    subtask_m = np.zeros((number_hidden,1))
    sizes[0] = 0
    output=[[[0]],[[0]],[[0]]]

    for bat in range(0,2):
        x_b = x[:,bat*b_size:(bat+1)*b_size]
        for i in range(0,nn_number_layer):
            output[i] = sigm(np.dot(nn_W[i],x_b) + np.tile(nn_b[i], (1,
b_size)))
            x_b = output[i]
            subtask_m[sum(sizes[0:i+1]):sum(sizes[0:i+2])] = subtask_m[
sum(sizes[0:i+1]):sum(sizes[0:i+2])] + np.expand_dims(g_f(output[i],
alph), axis=1)

    s_m = np.sort(subtask_m,0)
    s_index = np.argsort(subtask_m,0)
    subtask_index = s_index[:round(number_hidden*m_ratio)]
    return subtask_m, s_m, subtask_index
```

```
def put_W(nn,sizes,index,w_b):
    nn_number_layer= nn[0]
    nn_W= nn[1]
    nn_b= nn[2]
    nn_l2= nn[3]
    layer_number = 0

    for i in range(0,nn_number_layer):
       if (index < sizes[i+1]) or (index == sizes[i+1]):
            nn_W[layer_number][index,:] = w_b[0:-1].T
            nn_b[layer_number][index] = w_b[-1]
       else:
           index = index - sizes[i+1]

    nn=[]
    nn.extend((nn_number_layer,nn_W,nn_b, nn_l2))
    return nn


def get_maturity(x, nn, sizes,index):
    alph = 0.3
    m_ratio = 0.3
    nn_number_layer= nn[0]
    nn_W= nn[1]
    nn_b= nn[2]

    [m,_,_] = cal_maturity(x, nn_number_layer, nn_W, nn_b, sizes, alph,
m_ratio)
    maturity_value = m[index]
    return maturity_value


def fun_unsparse(train_x,train_y,nn, sizes,indival,sub_index):
    nn_w = put_W(nn,sizes,sub_index,indival)
    nn_number_layer= nn_w[0]
    nn_W= nn_w[1]
    nn_b= nn_w[2]
    nn_l2= nn_w[3]

    thert = 0.0001
    c = cal_cost(train_x,train_y,nn_number_layer, nn_W, nn_b, nn_l2)
    m = get_maturity(train_x,nn_w, sizes,sub_index)
    f_n = 1/c + thert*m
    return f_n


def DE_unsparse(nn,sizes,train_x,train_y,pop_1,sub_index):
    number_batch = 400
    m = np.size(train_x, axis=1)
    kk = np.random.permutation(range(m))
    x = train_x[:,kk[0:number_batch]]
    y = train_y[:,kk[0:number_batch]]

    [w_d,N] = pop_1.shape
```

```
F0 = 0.5
Np = N
CR = 0.9

value = np.zeros((Np,1))

Gm = 50
Gmin = np.zeros((1,Gm))
XG_next_1 = np.zeros((w_d,Np))
XG_next = np.zeros((w_d,Np))

G = 0
while (G < Gm):
    print(G,Gm,"1")
    for i in range(0,Np):
        a = 1
        b = Np
        dx = np.random.permutation(range(b-a+1)) + a - 1
        j = dx[0]
        k = dx[1]
        p = dx[2]
        if j == i:
            j  = dx[3]
            if k == i:
                 k = dx[3]
                 if p == i:
                       p = dx[3]

        suanzi = np.exp(1-Gm/(Gm + G))
        F = F0*2**suanzi
        son_1 = pop_1[:,p] + F*(pop_1[:,j] - pop_1[:,k])
        XG_next_1[:,i] = son_1

    XG_next_2 = XG_next_1
    TT = (np.random.rand(a,Np)>CR).nonzero()
    XG_next_2[TT] = pop_1[TT]

    for i in range(0,Np):
        a = fun_unsparse(x,y,nn,sizes,XG_next_2[:,i],sub_index)
        b = fun_unsparse(x,y,nn,sizes,pop_1[:,i],sub_index)
        if a > b:
            XG_next[:,i] = XG_next_2[:,i]

            value[i] = a
        else:
            XG_next[:,i] = pop_1[:,i]

            value[i] = b

    value_min, num_min = value.min(0),value.argmin(0)
    Gmin[0,G] = value_min

    pop_1 = XG_next
    G = G+1
```

```
    best_vector_1 = pop_1[:,num_min]
    return best_vector_1


def saCCDE_all(train_x,train_y,nn,sizes,subtask_index_need):
    N_P = 20
    nn_number_layer= nn[0]
    nn_W= nn[1]
    nn_b= nn[2]
    nn_l2= nn[3]
    nn_ori = nn

    task_number = np.size(subtask_index_need, axis=1)
    cost_last = cal_cost(train_x,train_y, nn_number_layer, nn_W, nn_b,
nn_l2)
    Cost_DE = np.zeros((task_number,1))

    for t in range(0,task_number):
        h = 'the number of subtask is'
        w_b = get_b(nn_ori,sizes,subtask_index_need[t])
        pop_1 = initial_pop(w_b,N_P)

        best_vector_1 = DE_unsparse(nn_ori,sizes,train_x,train_y,pop_1,
subtask_index_need[t])
        nn_de= put_W(nn_ori,sizes,subtask_index_need[t],best_vector_1)

        nn_de_number_layer= nn_de[0]
        nn_de_W= nn_de[1]
        nn_de_b= nn_de[2]
        nn_de_12= nn_de[3]

        Cost_de = cal_cost(train_x,train_y, nn_de_number_layer, nn_de_W,
nn_de_b, nn_de_12)

        if Cost_de < cost_last:
            TTTT = 1
            nn = nn_de
            cost_last = Cost_de
            Cost_DE[t,0] = Cost_de
            nn_ori = nn
        else:
            Cost_DE[t,0] = cost_last
            nn_ori = nn

    return nn
```

EK 2

```
__main__.py
-----------

#Başarı değerlendirme yöntemleri

def succes_rate(y_pred,y_test):
    a, b= y_test.shape
    tn,tp,fn,fp= 0,0,0,0
    for i in range(a):
        for j in range(b):
            yp=y_pred[i,j]
            y=y_test[i,j]

            #Tahmin edilen değerin ve gerçek sınıf değerinin 1 olması yani
True-Possitive durumu
            if yp==1 and y==1:
                tp=tp+1;

            #Tahmin edilen değerin ve gerçek sınıf değerinin 0 olması yani
True-Negative durumu
            elif yp==0 and y==0:
                tn=tn+1;

            #Tahmin edilen değerin 0 ve gerçek sınıf değerinin 1 olması
yani False-Negative durumu
            elif yp==0 and y==1:
                fn=fn+1;

            #Tahmin edilen değerin 1 ve gerçek sınıf değerinin 0 olması
yani False-Possitive durumu
            elif yp==1 and y==0:
                fp=fp+1;

    #True-Possitive ve False-Possitive değerinin 0 olması durumunda başarı
yüzdesi belirlenmeyeceği için başarı yüzdesi doğrudan 0'dır
    if tp==0 and fp==0:
        f1=0

    #True-Possitive ve False-Negative değerinin 0 olması durumunda başarı
yüzdesi belirlenmeyeceği için başarı yüzdesi doğrudan 0'dır
    elif tp==0 and fn==0:
        f1=0
    else:
        precision= tp/(tp+fp)   #Kesinlik hesabı
        recall= tp/(tp+fn)      #Duyarlılık hesabı
```

```
 #Kesinlik ve Duyarlılık değerlerinin 0 olması durumunda başarı yüzdesi
belirlenemeyeceği için başarı yüzdesi doğrudan 0'dır
        if precision==0 and recall==0:
            f1=0
        else:
            #Başarı yüzdesi yani kesinlik ve doğruluk değerlerinin harmonik
ortalaması
            f1= 2*(precision*recall)/(precision+recall)*100

    print('Doğruluk olasılığı: %{} '.format(f1))
    print('Precision: %{} '.format(precision*100))
    print('Recall: %{} '.format(recall*100))

    return f1
```

EK 3

```
__main__.py
-----------

import pickle
import cv2
import numpy as np
from skimage import morphology
from PIL import Image

from functions import crack_detection

#Çatlak tespiti yapmak istenen görselin uzantısının girilmesi
path= "Aaaaaaa.jpg"

#Yönteme başlamadan hemen önceki zaman değeri
start_time = time.time()

#Sırasıyla, çatlak olmayan nesnelerden arındırılmış binary görsel,
#çatlak alanlarının konumlarının ve çatlak bölgesi tespit edilmiş
#görselin elde edilmesi
image1, cracknodes, image2= crack_detection(path)

#Yöntemin tespit için harcadığı süre
passed_time = time.time() - start_time

#Elde edilen görsellerin gösterilmesi
cv2.imshow("Image 1",image)
cv2.imshow("Image 2",image1)
```

```
functions.py
-----------

import cv2
import numpy as np
from skimage import morphology
import pickle
from PIL import Image

def connectivitymaps(img):
    threshold= cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 135, 18)

    if np.count_nonzero(threshold)>6000 or
np.count_nonzero(threshold)<400:
        threshold= cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 135, 5)

    erosion_w= cv2.bitwise_not(threshold)
    arr = erosion_w > 0
    cleaned = morphology.remove_small_objects(arr, min_size=50)
    cleaned1 = morphology.remove_small_holes(cleaned, 150)
    result= 1*cleaned1
    return result


def sigm(P):
    X = 1/(1+np.exp(-P))
    return X


def detect_crack(img,nn_W,nn_b):
    for i in range(0,2):
        img= sigm(np.dot(nn_W[i],img) + nn_b[i])

    y_pred= img.T
    y_pred=np.round(y_pred)
    if y_pred[0][0]==1 and y_pred[0][1]==0:
        result=True
    else:
        result=False

    return result







def removed_small_object(img,cleaned1,nn_W,nn_b):
    areaimg= (img.shape[0])*(img.shape[1])
```

```
    cleaned2=cleaned1.astype(np.uint8)*255
    contours, _ = cv2.findContours(cleaned2, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for c in contours:
        (x, y, w, h) = cv2.boundingRect(c)
        rect = cv2.minAreaRect(c)
        box = cv2.boxPoints(rect)
        box = np.int0(box)
        ax=((box[0][0]-box[1][0])**2+(box[0][1]-box[1][1])**2)**0.5
        ay=((box[0][0]-box[3][0])**2+(box[0][1]-box[3][1])**2)**0.5
        area2= (1/2)*(box[0][0]*box[1][1]+box[1][0]*box[2][1]
+box[2][0]*box[3][1]+box[3][0]*box[0][1]-box[1][0]*box[0][1]-
box[2][0]*box[1][1]-box[3][0]*box[2][1]-box[0][0]*box[3][1])
        rate= min(ax,ay)/max(ax,ay)
        if area2<(areaimg/512) and rate>0.25:
            cleaned2[y:y+h,x:x+w]=0

        return cleaned2


def limited_1(x,y,w,h,cleaned5,image,img,nn_W,nn_b,cracknodes):

    if w<=80 and h<=80:
        box= [int(y+h/2-40),int(y+h/2+40),int(x+w/2-40),int(x+w/2+40)]
        if x+w/2-40<0:
            box[2],box[3]=0,80
        elif x+w/2+40>=cleaned5.shape[1]:
            box[2],box[3]= cleaned5.shape[1]-80, cleaned5.shape[1]
        if y+h/2-40<0:
            box[0],box[1]=0,80
        elif y+h/2+40>=cleaned5.shape[0]:
            box[0],box[1]= cleaned5.shape[0]-80, cleaned5.shape[0]

        part_of_image=img[box[0]:box[1],box[2]:box[3]]
        binary_image=connectivitymaps(part_of_image)
        line_image= binary_image.reshape(binary_image.shape[0]*
binary_image.shape[1],1)
        result= detect_crack(line_image,nn_W,nn_b)
        if result==True:
            cleaned_part=cleaned5[box[0]:box[1],box[2]:box[3]]
            contours_part, _ = cv2.findContours(cleaned_part,
cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
            if len(contours_part)<2:
                cv2.rectangle(image,(box[2],box[0]), (box[3],box[1]),
(255,255,255), -1)
                nodes=[box[0],box[1],box[2],box[3]]
                cracknodes.append(nodes)

    elif w>80 and h<=80:
        box= [int(y+h/2-40),int(y+h/2+40),x,x+w]
        abc=int(w/80)
        if int(w/80)/(w/80) != 1:
            fark=(int(w/80)+1)*80-w
```

```
            fark1,fark2=x,cleaned5.shape[1]-x-w
            if fark1>=fark/2 and fark2>=fark/2:
                x=x-int(fark/2)
            elif fark1<=fark/2 and fark2>=fark/2:
                x=0
            elif fark1>=fark/2 and fark2<=fark/2:
                x=cleaned5.shape[1]-(int(w/80)+1)*80
            abc= int(w/80)+1
        if y+h/2-40<0:
            box[0],box[1]=0,80
        elif y+h/2+40>=cleaned5.shape[0]:
            box[0],box[1]= cleaned5.shape[0]-80, cleaned5.shape[0]
        for i in range(abc):
            box1= [box[0],box[1],int(x+i*80),int(x+(i+1)*80)]
            part_of_image=img[box1[0]:box1[1],box1[2]:box1[3]]
            binary_image=connectivitymaps(part_of_image)
            line_image= binary_image.reshape(binary_image.shape[0]*
binary_image.shape[1], 1)
            result= detect_crack(line_image,nn_W,nn_b)
            if result==True:
                cv2.rectangle(image,(box1[2],box1[0]),(box1[3],box1[1]),
(255,255,255),-1)
                nodes=[box1[0],box1[1],box1[2],box1[3]]
                cracknodes.append(nodes)
            else:
                a=[(20,0),(-20,0),(0,20),(0,-20)]
                for i in range(4):
                    box2= box1[0]+a[i][0],box1[1]+ a[i][0],box1[2]+
a[i][1], box1[3]+a[i][1]
                    if min(box2)>=0 and box2[1]<=img.shape[0] and
box2[3]<=img.shape[1]:
                        part_of_image=img[box2[0]:box2[1],
box2[2]:box2[3]]
                        binary_image= connectivitymaps(part_of_image)
                        line_image= binary_image.reshape(
binary_image.shape[0]*binary_image.shape[1],1)
                        result= detect_crack(line_image,nn_W,nn_b)
                        if result==True:
                            cv2.rectangle(image,(box2[2],box2[0]),
(box2[3],box2[1]),(255,255,255),-1)
                            nodes=[box2[0],box2[1],box2[2],box2[3]]
                            cracknodes.append(nodes)
                            break

    elif w<=80 and h>80:
        box= [y,y+h,int(x+w/2-40),int(x+w/2+40)]
        abc=int(h/80)
        if int(h/80)/(h/80) != 1:
            fark=(int(h/80)+1)*80-h
            fark1,fark2=y,cleaned5.shape[0]-y-h
            if fark1>=fark/2 and fark2>=fark/2:
                y=y-int(fark/2)
            elif fark1<=fark/2 and fark2>=fark/2:
                y=0
```

```
            elif fark1>=fark/2 and fark2<=fark/2:
                y=cleaned5.shape[0]-(int(h/80)+1)*80
            abc= int(h/80)+1
        if x+w/2-40<0:
            box[2],box[3]=0,80
        elif x+w/2+40>=cleaned5.shape[1]:
            box[2],box[3]= cleaned5.shape[1]-80, cleaned5.shape[1]
        for i in range(abc):
            box1= [int(y+i*80),int(y+(i+1)*80),box[2],box[3]]
            part_of_image=img[box1[0]:box1[1],box1[2]:box1[3]]
            binary_image=connectivitymaps(part_of_image)
            line_image= binary_image.reshape(binary_image.shape[0]*
binary_image.shape[1],1)
            result= detect_crack(line_image,nn_W,nn_b)
            if result==True:
                cv2.rectangle(image,(box1[2],box1[0]),(box1[3],box1[1]),
(255,255,255),-1)
                nodes=[box1[0],box1[1],box1[2],box1[3]]
                cracknodes.append(nodes)
            else:
                a=[(20,0),(-20,0),(0,20),(0,-20)]
                for i in range(4):
                    box2= box1[0]+a[i][0],box1[1]+a[i][0],box1[2]+
a[i][1], box1[3]+a[i][1]
                    if min(box2)>=0 and box2[1]<=img.shape[0] and
box2[3]<=img.shape[1]:
                        part_of_image=img[box2[0]:box2[1],
box2[2]:box2[3]]
                        binary_image=connectivitymaps(part_of_image)
                        line_image= binary_image.reshape(
binary_image.shape[0]*binary_image.shape[1],1)
                        result= detect_crack(line_image,nn_W,nn_b)
                        if result==True:
                            cv2.rectangle(image,(box2[2], box2[0]),
(box2[3],box2[1]),(255,255,255),-1)
                            nodes=[box2[0],box2[1],box2[2],box2[3]]
                            cracknodes.append(nodes)
                            break

    return image


def crack_detection(path):

    ws_file= open(r"nn_W.pkl","rb")
    nn_W = pickle.load(ws_file)
    ws_file.close()

    bs_file= open(r"nn_b.pkl","rb")
    nn_b = pickle.load(bs_file)
    bs_file.close()

    img= cv2.imread(path,0)
    img_r= cv2.imread(path)
```

```
    threshold= cv2.adaptiveThreshold(img, 255,
cv2.ADAPTIVE_THRESH_GAUSSIAN_C, cv2.THRESH_BINARY, 135,
0.233*(img.shape[0]*img.shape[1])**0.33)

    erosion_w= cv2.bitwise_not(threshold)
    arr = erosion_w > 0

    cleaned = morphology.remove_small_objects(arr, min_size=50)
    cleaned1 = morphology.remove_small_holes(cleaned, 150)

    cleaned3=removed_small_object(img,cleaned1,nn_W,nn_b)
    cleaned5=cleaned3.copy()

    image= np.zeros((img.shape[0],img.shape[1]))
    areaimg= (img.shape[0])*(img.shape[1])

    contours, _ = cv2.findContours(cleaned5, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for c in contours:
        list1 = c.tolist()
        xlist=[]
        ylist=[]
        for i in range(len(c)):
            xlist= np.append(xlist,list1[i][0][0])
            ylist= np.append(ylist,list1[i][0][1])

        min_x = list1[xlist.argmin(0)][0]
        max_x = list1[xlist.argmax(0)][0]

        min_y = list1[ylist.argmin(0)][0]
        max_y = list1[ylist.argmax(0)][0]

        box=[min_x,max_x,min_y,max_y]
        for i in range(4):
            if box[i][1]>=10 and box[i][1]<img.shape[0]-10 and
box[i][0]>=10 and box[i][0]+10<img.shape[1]-10:
                part= cleaned5[box[i][1]-10:box[i][1]+10,box[i][0]-
10:box[i][0]+10]
                contours1, _ = cv2.findContours(part, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
                if len(contours1)==2:
                    array_of_tuples = map(tuple, contours1[0][0])
                    start=tuple(array_of_tuples)
                    startx,starty = start[0]

                    array_of_tuples = map(tuple, contours1[1][-1])
                    end = tuple(array_of_tuples)
                    endx,endy = end[0]
                    cv2.line(cleaned5,(startx+box[i][0]-10, starty+
box[i][1]-10),(endx+box[i][0]-10,endy+box[i][1]-10),[255,255,255],2)


    cleaned6=cleaned5.copy()
```

```
    arr = cleaned6 > 0
    cleaned6 = morphology.remove_small_objects(arr, min_size=50)
    cleaned6 = morphology.remove_small_holes(cleaned6, 150)*255
    cleaned6=cleaned6.astype(np.uint8)

    contours, _ = cv2.findContours(cleaned6, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for c in contours:
        (x, y, w, h) = cv2.boundingRect(c)
        rect = cv2.minAreaRect(c)
        box = cv2.boxPoints(rect)
        box = np.int0(box)
        ax=((box[0][0]-box[1][0])**2+(box[0][1]-box[1][1])**2)**0.5
        ay=((box[0][0]-box[3][0])**2+(box[0][1]-box[3][1])**2)**0.5
        area2= (1/2)*(box[0][0]*box[1][1]+box[1][0]*box[2][1]+
box[2][0]*box[3][1]+box[3][0]*box[0][1]-box[1][0]*box[0][1]-
box[2][0]*box[1][1]-box[3][0]*box[2][1]-box[0][0]*box[3][1])
        rate= min(ax,ay)/max(ax,ay)
        if area2<(areaimg/615):
            cleaned6[y:y+h,x:x+w]=0
        elif (areaimg/615)<area2<(areaimg/307) and rate>0.4:
            cleaned6[y:y+h,x:x+w]=0

    cracknodes=[]
    contours4, _ = cv2.findContours(cleaned6, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
    for c in contours4:
        (x, y, w, h) = cv2.boundingRect(c)
        cleaned_s= np.zeros((img.shape[0],img.shape[1]))
        cleaned_s=cleaned_s.astype(np.uint8)
        finished=False
        cleaned_s[y:y+h,x:x+w]=cleaned6[y:y+h,x:x+w]
        while finished==False:
            (x, y, w, h) = cv2.boundingRect(c)
            if w>80 and h>80:
                hull = cv2.convexHull(c,returnPoints = False)
                defects = cv2.convexityDefects(c,hull)
                far_list=[]
                for i in range(defects.shape[0]):
                    s,e,f,d = defects[i,0]
                    far = tuple(c[f][0])
                    far_list.append(far)

                list_i=[]
                box_list =[]
                for far in far_list:
                    xfar,yfar= far
                    if xfar<40:
                        xfar=40
                    elif xfar>cleaned_s.shape[1]-40:
                        xfar=cleaned_s.shape[1]-40
                    if yfar<40:
                        yfar=40
```

```
                    elif yfar>cleaned_s.shape[0]-40:
                        yfar=cleaned_s.shape[0]-40

                    #dikey
                    i_d=0
                    for far1 in far_list:
                        if xfar-40<=far1[0]<xfar+40:
                            i_d+=1
                    #yatay
                    i_y=0
                    for far2 in far_list:
                        if yfar-40<=far2[1]<yfar+40:
                            i_y+=1
                    if i_d>i_y:
                        boxdy=[xfar-40,y,xfar+40,y+h]
                        list_i.append(i_d)

                    else:
                        boxdy=[x,yfar-40,x+w,yfar+40]
                        list_i.append(i_y)

                    box_list.append(boxdy)

                value_max, num_max =
np.array(list_i).max(0),np.array(list_i).argmax(0)
                x1,y1,x2,y2=box_list[num_max]
                cv2.rectangle(cleaned_s,(x1,y1),(x2,y2),(0,0,0),-1)
                image=limited_1(x1,y1,x2-x1,y2-
y1,cleaned_s,image,img,nn_W,nn_b,cracknodes)

                arr = cleaned_s > 0
                cleaned_s = morphology.remove_small_objects(arr,
min_size=30)
                cleaned_s=cleaned_s.astype(np.uint8)*255
                contoursdy, _ = cv2.findContours(cleaned_s,
cv2.RETR_TREE, cv2.CHAIN_APPROX_NONE)
                if len(contoursdy)==0:
                    finished=True
                elif len(contoursdy)==1:
                    c=contoursdy[0]
                else:
                    breaak=True
                    for cnt in contoursdy:
                        (x, y, w, h) = cv2.boundingRect(cnt)
                        if w>80 and h>80:
                            breaak= False
                            c=cnt
                        else:
                            image=limited_1(x,y,w,h,cleaned_s,image,img,
nn_W,nn_b,cracknodes)

                    if breaak==True:
                        finished=True
```

```
            else:
                image=limited_1(x,y,w,h,cleaned_s,image,img,nn_W,nn_b,
cracknodes)
                finished=True
    image=image.astype(np.uint8)
    image_r= cv2.bitwise_not(image)
    edge=cv2.Canny(image,100,255)

    kernel = np.ones((2,2), np.uint8)
    img_dilation = cv2.dilate(edge, kernel, iterations=1)

    edge_r= cv2.bitwise_not(img_dilation)

    red_img  = np.full((img_r.shape[0],img_r.shape[1],img_r.shape[2]),
(0,0,255), np.uint8)
    white_img  = np.full((img_r.shape[0],img_r.shape[1],img_r.shape[2]),
(0,0,0), np.uint8)
    fused_img  = cv2.addWeighted(img_r, 0.8, red_img, 0.2, 0)

    crack_part= cv2.bitwise_and(fused_img,fused_img,mask=image)
    non_crack_part= cv2.bitwise_and(img_r,img_r,mask=image_r)
    all_picture_v1=crack_part+non_crack_part

    all_picture_v2=cv2.bitwise_and(all_picture_v1, all_picture_v1,
mask=edge_r)

    return cleaned6, cracknodes, all_picture_v2
```

EK 4

```
__main__.py
-----------

import glob
from crack_detection_new_v1 import crack_detection
import cv2
import time
import pickle
from math import floor
import numpy as np

#Manuel olarak belirlenmiş binary çatlak görselinin uzantısı
groundtruth_path= r"Aaaaaa.jpg"
#Başarı testi yapılacak renkli görselin uzantısı
rgb_image_path= r"Aaaaaa.jpg"

# GYBE_DSA ağırlık parametrelerinin içeriye aktarılması
ws_file= open(r"nn_W.pkl","rb")
nn_W = pickle.load(ws_file)
ws_file.close()

bs_file= open(r"nn_b.pkl","rb")
nn_b = pickle.load(bs_file)
bs_file.close()

TP=0    #True-Positive
FP=0    #False-Positive
FN=0    #False-Negative

#Başarı tespiti yapılacak görselin gri tonda içeriye aktarılması
img= cv2.imread(original_paths[i],0)

#Başarı tespiti yapılacak görselin renkli tonda içeriye aktarılması
img_r= cv2.imread(original_paths[i])

#Çatlak tespiti yönteminin uygulanması
image,cracknodes, image1= crack_detection(img,img_r,nn_W,nn_b)

#Ground truth çatlak görselinin içeriye aktarılması ve binary hale
getirilmesi
real_image= cv2.imread(real_paths[i],0)
_, real_image=cv2.threshold(real_image, 100, 255,cv2.THRESH_BINARY)
img_Real=real_image.copy()

#Çatlak görselindeki TP ve FP değerlerinin hesaplanması
for i in range(len(cracknodes)):
    area=real_image[cracknodes[i][0]:cracknodes[i][1],
cracknodes[i][2]:cracknodes[i][3]]
```

```
    img_Real[cracknodes[i][0]:cracknodes[i][1],
cracknodes[i][2]:cracknodes[i][3]]=0
    if np.count_nonzero(area)>0:
        TP=TP+1
    else:
        FP=FP+1

#Çatlak görselindeki FN değerinin hesaplanması
_, contours, _ = cv2.findContours(img_Real, cv2.RETR_TREE,
cv2.CHAIN_APPROX_SIMPLE)
for c in contours:
    (x, y, w, h) = cv2.boundingRect(c)
    ratio=w*h/(80*80)
    number=floor(ratio)+1
    FN=FN+number

precision= TP/(TP+FP)
recall= TP/(TP+FN)
f1= 2*(precision*recall)/(precision+recall)
print("Kesinlik: %"+ precision*100, "Duyarlılık: %"+ recall*100, "F1-
Score: %"+ f1*100)
```