T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR BİLİMLERİ VE MÜHENDİSLİĞİ BİLİM DALI

DERİN ÖĞRENME KULLANILARAK SAHTE PLAKALI ARAÇ TESPİT SİSTEMİ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Burak AĞGÜL

İstanbul
Şubat-2021

T.C.
İSTANBUL SABAHATTİN ZAİM ÜNİVERSİTESİ
LİSANSÜSTÜ EĞİTİM ENSTİTÜSÜ
BİLGİSAYAR MÜHENDİSLİĞİ ANABİLİM DALI
BİLGİSAYAR BİLİMLERİ VE MÜHENDİSLİĞİ BİLİM DALI

DERİN ÖĞRENME KULLANILARAK SAHTE PLAKALI ARAÇ TESPİT SİSTEMİ GELİŞTİRİLMESİ

YÜKSEK LİSANS TEZİ

Burak AĞGÜL

Tez Danışmanı
Dr. Öğr. Üyesi Gökhan ERDEMİR

İstanbul
Şubat-2021

TEZ ONAYI

Lisansüstü Eğitim Enstitüsü Müdürlüğüne,

Bu çalışma, jürimiz tarafından Bilgisayar Mühendisliği Anabilim Dalı, Bilgisayar Bilimleri ve Bilgisayar Mühendisliği Bilim Dalında YÜKSEK LİSANS TEZİ olarak kabul edilmiştir.

Danışman Dr. Öğr. Üyesi Gökhan Erdemir

Üye Dr. Öğr. Üyesi Aydın Tarık Zengin

Üye Prof. Dr. Tahir Çetin Akıncı

Onay

Yukarıdaki imzaların, adı geçen öğretim üyelerine ait olduğunu onaylarım.

Prof. Dr. Ali Güneş
Enstitü Müdürü

BİLİMSEL ETİK BİLDİRİMİ

Yüksek lisans tezi olarak hazırladığım "**Derin Öğrenme Kullanılarak Sahte Plakalı Araç Tespit Sistemi Geliştirilmesi"** adlı çalışmanın öneri aşamasından sonuçlandığı aşamaya kadar geçen süreçte bilimsel etiğe ve akademik kurallara özenle uyduğumu, tez içindeki tüm bilgileri bilimsel ahlak ve gelenek çerçevesinde elde ettiğimi, tez yazım kurallarına uygun olarak hazırladığımı, bu çalışmamda doğrudan veya dolaylı olarak yaptığım her alıntıya kaynak gösterdiğimi ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu beyan ederim.

Burak AĞGÜL

ÖNSÖZ

Araştırmamdaki her aşamada bana yardımcı olan değerli tez danışmanım Dr. Öğr. Üyesi Gökhan ERDEMİR' e, lisans ve yüksek lisans eğitimim boyunca benden desteklerini esirgemeyen sevgili aileme teşekkürlerimi sunarım.

Burak AĞGÜL

İstanbul-2021

ÖZET

DERİN ÖĞRENME KULLANILARAK SAHTE PLAKALI ARAÇ TESPİT SİSTEMİ GELİŞTİRİLMESİ

Burak AĞGÜL
Yüksek Lisans, Bilgisayar Bilimleri ve Mühendisliği
Tez danışmanı: Dr. Öğr. Üyesi Gökhan ERDEMİR
Şubat-2021,92 Sayfa

Bu çalışmada, sahte plakalı araçları tespit edebilmek için araçların marka, model, renk, plaka gibi özelliklerini derin öğrenme yöntemi kullanılarak karşılaştıran ve tespit eden bir sistem tasarlanmıştır. Çalışmanın amacı, günlük hayatta aktif olarak kullanılan araçların özelliklerini kontrol ederek mevcut sistemlere ek olarak yardımcı bir sahte plaka tespit sistemi geliştirmektir. İlgili devlet kurumları tüm motorlu taşıtlara ait ayrıntılı tüm bilgileri kendi veri tabanında saklamakla sorumlu olduğu bilinmektedir. Rastgele seçilen veya başka bir araca ait sahte araç plakası marka, model, renk gibi tescil kayıtlarından farklıdır. Genelde sahte plakaya sahip araçlar yasa dışı ve cezai eylemlerde kullanılmaktadır. Bu nedenle, sahte araçları tespit etmek büyük önem taşımaktadır. Genel olarak, sahte plakaları tespit etmek için plaka tanıma sistemleri kullanılır. Bu yaklaşım, rastgele seçilen plaka numaraları için kullanışlıdır. Genelde güvenlik birimleri tarafından kullanılan bu tür sistemler, ücretli geçiş yolları, köprü geçişleri, otopark giriş-çıkışları, siteler, gümrük kapıları vb. İçin de kullanılmaktadır. Ancak çalıntı plakalı araçlarda kullanışlı değildir. Bunu tespit edebilmek için araç markasını, modelini, rengini ve plaka numarasını karşılaştırabilecek bir sistemin bulunması gerekmektedir. Bu çalışmada, sahte araç plaka algılama sistemi, araç marka, model, renk ve plaka numaralarının derin öğrenme kullanılarak karşılaştırılmasına dayalı olarak geliştirilmiştir. Tasarlanan bu sistemi diğerlerinden ayıran en temel fark veri setinin sıfırdan oluşturulmasıdır. Oluşturulan veri seti üzerinde eğitim ve test için derin öğrenme yöntemi kullanılarak farklı aktivasyon fonksiyonları ve optimizasyon yöntemleri ile irdelenmiş en iyi sonucu veren fonksiyonlar ve optimizasyon parametreleri sonuçları ile birlikte gösterilmiştir. Sistem sanal olarak oluşturulan bir veri tabanı ile test edilmiştir.

Anahtar Kelimeler: Araç marka/model, renk, plaka, Derin Öğrenme, Konvolüsyonel Sinir Ağı (CNN), Keras

ABSTRACT

DEVELOPMENT OF A COUNTERFEIT VEHİCLE LICENSE PLATE DETECTION SYSTEM BY USING DEEP LEARNING

Burak AĞGÜL
Master, Computer Science and Engineering
Thesis Advisor: Asst. Prof. Dr. Gökhan Erdemir
February-2021,92 Pages

In this study, the detection of the counterfeit vehicle license plate was designed by taking into consideration the characteristics of the vehicles such as brand, model, color, and license plate by using deep learning. The main aim of the study is to develop an additional supplementary system to check the specifications of the vehicles that are actively used in daily life to prove real or counterfeit. It is known that the relevant government department is responsible to store information about details of all motor vehicles in traffic in their database. Counterfeit vehicles' license plate which is randomly chosen or belonging to another vehicle is different from registration records such as brand, model, and color. In general, vehicles that have the counterfeit license plate are used in illegal and criminal acts. Hence, it is crucially important to detect counterfeit vehicles. In general, license plate recognition systems are used to detect counterfeit license plates. This approach is useful for randomly chose plate numbers. These kinds of systems, which are generally used by security units, are also used for the toll-pass roads, bridge crossings, parking lot entry-exit, sites, customs gates, etc. But, it is not useful on stolen license plates that exist in records. It is necessary to develop a system that can compare vehicle brand, model, color, and license plate number. In this study, the counterfeit vehicle license plate detection system has been developed based on comparing vehicle brand, model, color, and license plate number by using deep learning. The database which is used in this study was completely done over again. Using the deep learning method for training and testing on the data set, the functions and optimization parameters that give the best results examined with different activation functions and optimization methods are shown together with their results. The system has been tested with a virtually created database.

Key Words: Vehicle, brand, model, color, licence plate, deep learning, Convolutional Neural Network (CNN), Keras

İÇİNDEKİLER

TABLO LİSTESİ

ŞEKİLLER LİSTESİ

BİRİNCİ BÖLÜM

İKİNCİ BÖLÜM

ÜÇÜNCÜ BÖLÜM

DÖRDÜNCÜ BÖLÜM

BEŞİNCİ BÖLÜM

KISALTMALAR LİSTESİ

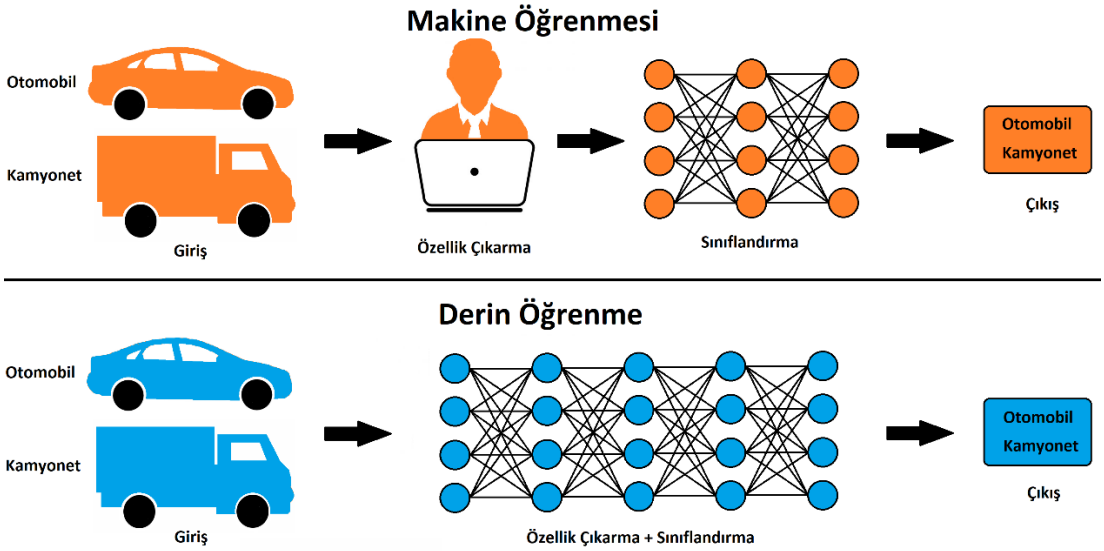
GPU	: Grafik İşleme Birimi (Graphics Processing Unit)
TPU	: Tensör İşleme Birimi (Tensor Processing Unit)
CPU	: Merkezi İşleme Birimi (Central Processing Unit)
PIL	: Python Görüntü Kütüphanesi (Python Image Library)
YSA	: Yapay Sinir Ağı
CNN	: Konvolüsyonel Sinir Ağları (Convolutional Neural Networks)
RNN	: Tekrarlayan Sinir Ağı (Recurrent Neural Networks)
LSTM	: Uzun Kısa Dönemli Bellek (Long Short Term Memory)
RBM	: Kısıtlanmış Boltzmann Makineleri (Restricted Boltzmann Machines)
DBN	: Derin İnanç Ağları (Deep Belief Networks)
DAE	: Derin Oto Kodlayıcılar (Denoising Autoencoders)
PTS	: Plaka Tanıma Sistemi
LMM	: Levenberg-Marquardt Metodu
CMD	: Komut Sistemi
IDE	: Entegre Geliştirme Ortamı (Integrated Development Environment)
CV	: Bilgisayarla Görme (Computer Vision)
MLL	: Makine Öğrenme Kütüphanesi (Machine Learning Library)
DB4S	: Veri Tabanı for Sqlite (Database for Sqlite)
RGB	: Kırmızı Yeşil Mavi renkleri (Red Green Blue)
BTAPTS	: Bilgisayar Tabanlı Araç Plaka Tanıma Sistemi
OpenCV	: Açık Kaynaklı Bilgisayarlı Görüntü İşleme (Open Source Computer Vision)
API	: Uygulama Programlama Arayüzü (Application Programming Interface)

BİRİNCİ BÖLÜM
GİRİŞ

Son yıllarda trafikte seyir halindeki araç sayısındaki artış bazı güvenlik önlemlerindeki stratejilerinin geliştirilmesinde en temel faktör olmuştur. Bu tarz adımların genellikle önlem olarak adlandırılmasının sebebi hayatımızda büyük öneme sahip bu araçların kontrolünün sağlanmasıdır. Örneklendirmek gerekirse hırsızlık, gasp vb. kaçış durumları, otopark giriş-çıkış, köprü ve otoyol geçişleri esnasında kontrolü sağlayabilmek için seyir halindeki tüm motorlu araçların belirli bilgilerinin tutulduğu veri tabanı üzerinden kontrol edilmesidir. Daha birçok eklenebilecek olumlu ya da olumsuz durumlarda ihtiyaç duyulan bu tarz sistemlerin çalışma prensibi araçların belirli özellikleri üzerine kuruludur. Temel olarak araçların model, model yılı, renk ve plaka vb. ayırt edici özellikleri kullanılır.

Gelişen teknoloji ile birlikte bu tarz sistemler üzerine farklı çalışmalar sürekli olarak yapılmaktadır. Araç görüntüsünden anlık olarak model, renk ve plaka gibi bilgilerin hızlı ve doğru bir şekilde tespit edilmesinin amaçlanması bu tarz sistemlerin sürekli geliştirilmesi üzerine çalışmalar yapılmasına sebep olmuştur. Günümüz teknolojisinde bu tarz sistemler artık derin öğrenme yöntemleri kullanılarak tasarlanıp daha hızlı ve daha doğru sonuçlar veren sistemler olarak geliştirilmiştir.

Derin Öğrenme, derin sinir ağlarını ifade eden bir makine öğrenme yöntemidir. Verilen bir veri kümesini yapay sinir ağlarını kullanarak çıktıları tahmin edecek yapay zekâyı eğitmemize imkân veren bir sistemdir. Derin öğrenmenin uygulanabilmesi için çok fazla veriye ihtiyaç duyulmaktadır. Aslında derin öğrenme makine öğrenme yönteminin daha gelişmiş şeklidir. En temel fark veri içerisinden kendi kendine öğrenerek yeni özellikler oluşturmasıdır. Belirli özelliklerin makine öğrenmesinde kullanıcılar tarafından doğru bir şekilde tanımlanması ve oluşturulması gerekirken derin öğrenmede bu özellikleri sistemin üretip etiketlemesi ve sonucunda yapay sinir ağlarını kullanarak bir çıktı üretmesi beklenir (A Şeker et al., 2017).



Şekil 1.1: Makine Öğrenmesi ile Derin Öğrenmenin Farkı

Günümüzde donanım seviyeleri arttığı için derin öğrenmede daha çok çalışmalara yapılmaya başlanmıştır. Çünkü derin öğrenme güçlü GPU(Grafik İşleme Birimi) isteyen bir yapıdır. Gelişen teknoloji ile birlikte derin öğrenme yöntemlerinin kullanıldığı alanlarda çeşitlilik kazanmıştır. Örneğin sağlık alanından tutun, günlük hayatta sıklıkla kullandığımız akıllı telefonlar dahil birçok alanda kullanılır (Kaya et al., 2019). Derin öğrenme yöntemlerinin sıklıkla kullanıldığı bazı alanlardan bahsetmek gerekirse, bu alanlar şu şekildedir. Yüz tanıma, ses tanıma, otonom sistemler, savunma ve güvenlik, sağlık vb. alanlardır. Birkaç örnekle uygulama alanlarından söz edersek; (Yağcı et al., 2005).

Yüz tanıma için akıllı telefonlarda kullanılan kilit ekranının açılması ya da insan topluluğun fazla olduğu yerde kamera yardımıyla alınan görüntüden kişi tespiti gibi uygulama alanlarında kullanılır (Yıldız et al., 2019).

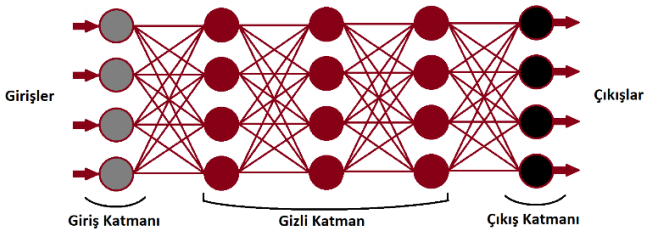
Ses tanıma için telefon bankacılığında kullanılan yaygın bir uygulamadır. Örnek olarak operatörleri aradığımızda operatörüm beni sesimden tanır diyerek belli başlı işlemleri yapabilme imkânı verir (Miao et al., 2016).

Otonom sistemler için genellikle araçlarda kullanılan otomatik pilot özelliği ön plana çıkan bir sistemdir. Trafikte seyir halindeyken ani şerit değiştirme ya da önündeki araç ile mesafe azalması durumunda otomatik karar verme ve uygulama yeteneğine sahip sistemler (Huval et al., 2015).

Savunma ve güvenlik alanlarında kullanım için örnek olarak güvenlik güçlerinin kullandığı kameralarda ekrana yansıyan nesneler için hangi tür canlı olduğunu görünmesi (genellikle gece için) gösteren ve gerektiğinde konuma yaklaştığında alarm veren sistemler (Yağcı et al., 2005) , (Carlson, 2019).

Sağlık alanı için genellikle kanser tedavisinde kullanılır. Kanserli hücre görüntülerinin öğretildiği sistemin, yeni görüntüler üzerinde kanserli hücre olup olmadığı çıkarımını sağlar (Kaya et al., 2019).

Bu bahsettiğimiz alanlardan bazıları bundan 10 yıl önce de kullanılıyordu. Örneğin eskiden kullanılan yüz tanıma sistemi ile günümüzde kullanılan yüz tanıma sistemlerinde ciddi farklar vardır. Bunun en temel sebebi günümüzde artık çok daha güçlü GPU'lar ile çalışılıyor olmasıdır. Aynı şekilde kullanılan veri setlerinin artmasında, derin öğrenme alanın genişlemesine sebebiyet vermiştir. Ancak bu kadar veri setinin işlenip yeni veriler doğurabilmesi için yalnızca doğru bir sınıflandırma gerektirir. Bu işlemi gerçekleştirmek için insan beynine benzeyen bir yapıda olan yapay sinir ağına ihtiyaç duyulur. Bu sinir ağı insan beynindeki gibi birbirine bağlı nöronlardan oluşmaktadır. Bir sinir ağında Giriş, Gizli ve Çıkış Katmanı olmak üzere 3 katman vardır. Giriş katmanında veri seti girdisi, gizli katmanda matematiksel işlemler ve çıkış katmanında sonuç bilgisi verilir.



Şekil 1.2: Temel Yapay Sinir Ağı Yapısı

1980'li yıllarda yapay sinir ağları kullanılmaya başlanmıştı (Öztürk & Şahin, 2018). Fakat derin öğrenme CNN kullanımı ile daha başarılı bir hale gelmiştir . Tezimizde de kullanılan bu yaklaşımın temel mantığı derin öğrenmenin giriş, gizli ve çıkış katmanlarından oluşan bir yapısının olmasıdır. Bu katmanlar sayesinde bilgisayarı daha fazla besleyerek öğrenmesine yarayan bir algoritma olması ile ön plana çıkıyor.

1.1. Derin Öğrenme Mimarileri

Derin öğrenme yöntemlerinde yaklaşık 20 adet sınıf bulunmaktadır. Fakat bu sınıflar temel olarak 6 sınıftan türetilen yöntemlerdir. Türetmekten kasıt olarak örnek verecek olursak bir yöntem içindeki katman sayılarını artırarak daha kompleks yapı oluşturulması ve bunu yeni bir yaklaşım olarak tanınmasıdır. Temel olarak anılan bu 6 yöntem kısaca alt bölümde açıklanmıştır.

1.1.1. Konvolüsyonel Sinir Ağları (Convolutional Neural Networks) (CNN)

Bir diğer adıyla evrişimsel sinir ağları olarak kullanılan bu yöntem bir görüntü üzerinde çeşitli objeleri veya nesneleri ayrıştırıp analiz yapmamıza olanak sağlayan bir algoritmadır. İleri yönlü bir sinir ağı olan CNN, çıkış olarak hayvanların görme merkezleri göz önüne alınarak ortaya konulmuş bir yöntemdir. Evrişimsel işlemler derken bir nöronun kendi uyarı alanından kendine bağlı uyaranlara verdiği cevap olarak değerlendirilebilir (Şeker, 2017), (Fukushima, 1980), (Hubel & Wiesel, 1968), (Lecun et al., 1998). CNN ilk kez Le Cun tarafından 1988 yılında ortaya çıkarılmış ve 1998 yılına kadar geliştirilmesi devam edilen bu yapay sinir ağı LeNet olarak bilinen mimaridir (Y, 1988). Evrişim olayı basitçe ifade edilirse, matematik, fizik, mühendislik vb. uygulamalarındaki karmaşık işlemleri basitleştirmek için kullanılan işlemlerdir.

1.1.2. Tekrarlayan Sinir Ağı (Recurrent Neural Network) (RNN)

Tekrarlayan sinir ağlarının yapısını daha kolay anlamak için basit bir örnek verebiliriz. Bizler düşüncelerimizi önceden öğrendiğimiz bilgilere dayanarak ortaya koyarız. Yani daha basitçe anlatmak gerekirse bir cümleyi okuduğumuzda her kelimesini bir önceki kelimeyi dikkate alarak anlarız ya da anlamlandırırız. Geleneksel sinir ağları bu şekilde çalışmaz. Dolayısıyla bu çok büyük bir eksikliktir. RNN ise bu tarz sorunları ele alan bir yapıya sahiptir. Tekrarlayan sinir ağı ilk olarak 1980’ lerde ortaya atıldığında Simple Recurrent Network adı verilmiştir. Daha sonra 1990’ larda Jeff Elman tarafından tasarlanan tekrarlayan sinir ağı önerilmiştir. Bu algoritmanın temel mantığı sıralı bilgiler kullanmaktır. Geleneksel sinir ağında tüm girdiler ile çıktıların birbirinden bağımsız olduğu varsayılır. Örnek olarak bir cümlede bir kelimeyi tahmin edebilmek için ondan önce hangi kelimenin geldiği

bilinmesi gerekir. RNN mimarisinin yinelenen olarak nitelendirilmesinin sebebi, bir dizinin her elemanı için ondan önceki çıktılara bağlı olarak hareket etmesidir. Jeff Elman tarafından tasarlanan algoritma ile bir cümlenin isim ve fiil kategorileri ayrılmış hatta isimler arasında canlı-cansız, insan-hayvan, hayvanlar arasında avcı-yırtıcı gibi kümeler oluşturulmuştur (Elman, 1990), (Şeker, 2017).

1.1.3. Uzun / Kısa Dönemli Bellek (Long / Short Term Memory) (LSTM)

RNN mimarisinin temelinde önceki bilgi kullanıma endeksli bir yapı vardır. Örneğin "Balık suda olur" cümlesinde "su" kelimesinin tahmin edilmesi kolaydır. Ancak bağlamlar arası mesafe arttığında RNN mimarisi önceki bilgileri kullanması zorlaşır. Örneğin, "Asya ile Avrupa' yı birbirine bağlıyorlar…Bu nedenle coğrafi öneme sahiptir bu köprüler." gibi bir metinde "köprü" kelimesini tahmin etmek istenildiğinde içinde bulunduğu cümleden yola çıkılarak bir bağlantı ya da köprü olacağını tahmin etmek kolaydır. Fakat doğru kelimenin köprü olduğunu tahmin edebilmek için cümlenin başını hafıza tutmak gerekir. Aslında LSTM ağlarının RNN ağlarından bir farkı yoktur. RNN' lerin dezavantajı olarak bilinen bağlam boşluklarının tahmin edilmesi ihtiyacını karşılayan bir yapı olarak ortaya çıkmıştır. Bu yapı içinde hafıza hücreleri yer alır. Ağ mimarisi içinde bulunan hafıza hücreleri içinde önceki durum ile girdi bilgisi tutarlar. Böylece hücreler hangi veriyi tutacağını ya da sileceğine karar verirler. Daha sonra önceki mevcut bellek ile giriş verisini birleştirerek aralarında bir uzun vadeli bağımlılıklarını ortadan kaldırarak veri dizilerini devam ettiriler (Hochreiter & Schmidhuber, 1997), (Şeker, 2017).

1.1.4. Kısıtlanmış Boltzmann Makineleri (Restricted Boltzmann Machines) (RBM)

Kısıtlanmış Boltzmann makinesi (RBM), Harmonium adıyla ilk olarak 1986 yılında ortaya çıkmıştır (Scholar & Smolensky, 1986). Fakat daha sonra 2006' da Geoffrey Hinton ve arkadaşları birlikte hızlı bir öğrenme algoritması olarak ortaya çıkarılmıştır (Hinton, 2006). RBM aslında Boltzmann Makinalarının bir çeşitidir. Görünür ve gizli noktalar olmak üzere birbirine bağlantılı graflardan oluşmaktadır (Salakhutdinov & Hinton, 2009). Sınıflandırma, regrasyon ve özellik öğrenimi işlemlerini yapan Boltzmann makinesi giriş veri seti üzerinden olasılıksal dağılımlarını öğrenebilen bir sinir ağıdır (Şeker, 2017).

1.1.5. Derin İnanç Ağları (Deep Belief Network)(DBN)

Geoffrey Hinton ve arkadaşları tarafından önerilen Derin inanç ağları (DBN), bir önceki bölümde bahsettiğimiz RBM' lerin yığını olarak tanımlanmaktadır. Derin inanç ağları Kısıtlı Boltzmann Makinelerinin (RBM) yığınlarıyla oluşturulur. RBM' ler sırasıyla eğitilerek öğrenilmesiyle gerçekleşir. Fakat yatayda katmanlar arasında bir iletişim yoktur. En son katman olarak softmax katmanı ile sınıflandırma ya da denetimsiz bir öğrenme için kümeleme yapan yapıya sahiptir (Hinton, 2006), (Şeker, 2017).

1.1.6. Derin Oto Kodlayıcılar (Denoising Autoencoders)(DAE)

Oto-kodlayıcılar (AE) denetimsiz öğrenme için kullanılan diabolo ağı olarak adlandırılan yapay sinir ağıdır (Bengio, 2009). AE, girdi katmanındaki değerleri çıktı katmanına kopyalan bir sinir ağı türüdür. Yani, sinir ağına girdi olarak verilen veriyi, çıktı katmanında tekrar oluşturur. AE, kısaca girdi verisinin sıkıştırılmış gösteriminden en iyi özelliklerin öğrenilmesini hedefleyen bir ileri beslemeli sinir ağıdır (Krizhevsky & Hinton, 2011). Derin oto-kodlayıcılar (DAE) ise her bir katmandaki çıktıların ardışık katmanın girişlerine bağlandığı AE' lerin çok katmanlarından oluşan sinir ağı olarak bilinirler (Şeker, 2017), (Abdulkadir & Yüksek, 2017).

1.2. Plaka Tanıma Sistemleri

Plaka tanıma sistemleri günümüzde araçların hız ihlali, ışık ihlalleri, ücretli otoyol geçişleri, gümrükler, avm otoparkları, otopark giriş-çıkış, site veya konut girişleri vb. yerlerde kullanılmaktadırlar. Temel olarak aracın ön ya da arka tarafından bir kamera ile alınan görüntü birkaç adımlık morfolojik işlemlerden geçtikten sonra plaka kısmı kırpılır. Kırpılan görüntü üzerinde daha önceden hazırlanmış birçok karakter üzerinde eğitilmiş ve test edilmiş karakter okuma ayrıştırma algoritması kullanılarak plaka okunur ve ilgili ekrana yazdırılır.

Plaka Tanıma Sistemi (PTS) üzerine yapılan çalışmalardan birkaç tanesi incelediğinde ilk olarak ele aldığımız Orhan Kaplan ve arkadaşları tarafından Erciyes Üniversitesi kampüs girişi için yapılmış olan Araç Plaka Tanıma Sistemi (APTS)' dir. Sistemin temel mantığı çok sade ve basit bir anlaşılırlığı vardır. Kampüse giren araçların kaydının tutulması ve daha güvenilir bir kampüs içi trafik amaçlanmıştır.

Kampüs girişine kurulan bir bariyer ve kamera ile girişe yaklaşan aracın 640x480 piksel çözünürlükte fotoğrafını çeken sistem daha sonra bu fotoğrafı sisteme bağlı olan bilgisayar üzerindeki program sayesinde algılar ve geçişe izin verir. Bu algılama işlemleri yapan algoritma ilk olarak alınan araç görüntüsüne Grey-Level Edge Detection adı verilen türkçesi gri-seviyeli kenar belirleme işlemi uygulanır. Bu morfolojik işlemler uygulanırken Canny, Prewitt, Sobel ve Zero Crossing gibi birçok kenar belirleme operatörleri kullanılmış fakat yeterli başarı oranı yakalanamamıştır. Bu durumda araç görüntüsü üzerinde ilk pikselden başlanılarak gezilen resim üzerinde beyaz piksel değerleri sayılmış ve en yoğun olduğu bölgenin bulunması amaçlanmıştır. Bulunan bölge için en ihtimalle plaka bölgesi olduğu kanaatine varılmıştır. Bulunan bölge kırpılarak fotoğraf üzerinde daha net karakter tanıma işlemi yapılabilmesi için bazı morfolojik işlemlerden geçirilmiş ve karakter tanıma da yapay sinir ağlarına ihtiyaç duyulmuştur (Kaplan et al., n.d.).

Bunun sebebi birçok literatürde karakter tanıma için Şablon Karşılaştırma, kritik nokta, hiyerarşik karakter tanıma vb. gibi methodlar uygulanmış fakat bu yöntemlerde benzer karakter iyi sonuçlar vermediği gözlemlenmiştir. Örneğin O harfi ile 0 sayısı veya D harfinin benzerliği, bir diğeri B harfi ile 8 sayısının benzerliği gibi karakterlerde başarılı sonuçlar alınamamıştır. Bunun önüne yapay sinir ağları ile geçilmeye çalışılmış ve başarı oranı yüksek seviyelerde sonuçlar alınmıştır. Yapay sinir ağları (YSA) son yıllarda birçok alanda karakter tanıma için kullanılmış ve başarı oranı yüksekliği kanıtlanmış yöntemlerdendir (Brown, 1992). Bu PTS sistemin karakter tanıma için kullanılan YSA' ları eğitmede Levenberg-Marquardt Metodu (LMM) kullanılmıştır (Yildiz et al., n.d.), (Sağiroğlu & Beşdok, 2012), (Kaplan et al., n.d.). Sistem 1.5 sn. içerisinde %99.25 başarı oranı ile çalışan, kötü hava şartlarında %97' lere gerileyen bir oranla çalıştığı gözlemlenmiştir.

İncelenen bir diğer çalışma ise Okan Bingöl ve arkadaşı tarafından hazırlanan Bilgisayar Tabanlı Araç Plaka Tanıma Sistemi (BTAPTS)' dir. Sistemin hazırlanış ve uygulanış şekli incelendiğinde plaka okunmasında Blob Coloring algoritması kullanılmış olup karakter tanımada ise hemen üstte handikaplarından bahsettiğimiz şablon eşleştirme yani template matching yöntemi kullanıldığı görülmüştür. Araç görüntüsünün elde edilmesi ve elde edilen görüntü üzerinde plaka bölgesi bulma

işlemleri aynı şekilde ilerliyor ve iki çalışmayı birbirinden ayıran temel noktaya gelindiğinde ise başarı oranlarının farklı olmasındaki en temel sebebin karakter tanıma olduğu gözlemleniyor. Karakter tanıma da şablon eşleştirme methodunun problemlerinden bahsetmiştik. Bazı karakterlerin benzer olması gibi durumlar ve bunların üstüne kötü hava şartlarından oluşan sebeplerden dolayı alınan görüntüde kırpılan plaka bölgesi tanınmak istediğinde karakterlerin doğru olarak görüntülenememesi gibi sorunlar meydana gelmiştir. Bütün bu bahsedilen sorunların üstesinden daha iyi gelebilmek için şu an kullanılan en güncel yöntemin yapay sinir ağlarının kullanılmasıdır. Aksi takdirde bu çalışmada olduğu gibi 800x600 piksel ölçülerinde daha büyük fotoğraflar alındığı halde sonuçların doğruluk oranı %87' lik bir oran ile kaldığı görülmüştür. Sistem üstte bahsettiğimiz çalışmaya göre 0,5 sn.' lik gecikme ile 2sn. sonuç üretiyor (Ngöl & Cu, 2008).

Son olarak incelediğimiz bu çalışma ise Kerim Kürşat Çevik ve arkadaşı tarafından hazırlanan Görüntü İşleme Yöntemleriyle Araç Plakalarının Tanınarak Kapı Kontrolünün Gerçekleştirilmesi' dir. İncelenen çalışma hemen üstte Okan Bingöl ve arkadaşlarının izlediği yol ile benzer olup plaka bölgesi bulma adımları benzer ve okuma algoritması Blob Coloring algoritması olmak üzere temel olarak benzerdir. Bu çalışmayı diğerinden ayıran tek fark karakter okuma da yapmış olduğu gürültü yok etme gibi iyileştirme işlemleri ile birlikte almış olduğu doğruluk başarı oranı %88,1 ile okunması durumudur. Sistemin çalışma süresinden de bahsedilmemiştir (Çevik & Çakır, 2010).

Bu çalışmada belirli araç marka modelleri, plaka bilgisi ve renk bilgisini tanıyan bir sistem oluşturulmuştur. Sistemin temel amacı derin öğrenme yöntemleri kullanılarak aracın marka model bilgisini bulabilmektir. Fakat çalışma da plaka ile renk bilgisi üzerinde ayrıntılı işlemler yapılmamış olup temel olarak bir algoritma ile tanınma işlemi yapılmıştır.

Derin öğrenme işlemleri bilindiği üzere karmaşık veya kompleks olarak adlandırdığımız işlemlerden oluştuğu için güçlü bilgisayarlara yani donanımlara ihtiyaç duyarlar. Bu donanım gereksinimleri için Google Colab' ın sunmuş olduğu sanal GPU kullanılmıştır. Colab ortamına veri seti aktarımı için aynı şirketin bir

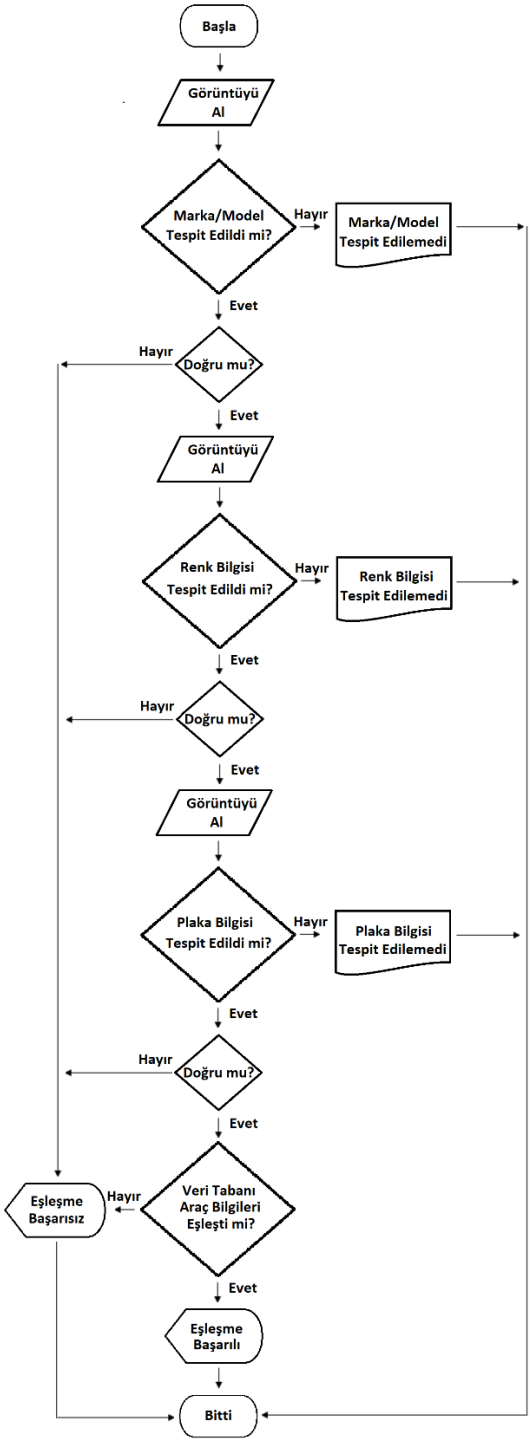
diğer hizmeti de olan Drive uygulaması kullanılmıştır. Hali hazırda bulunan eğitilmiş veri setleri ya da veri seti sonuçlarını kullanmak yerine kendi oluşturduğumuz veri seti kullanılarak sıfırdan eğitim-test-gerçekleme işlemi yapılmıştır. Bu esnada farklı parametreler farklı değerler ile maksimum seviyede hız ve doğruluk oranı için en uygun değerler elde edilerek uygulanmıştır.

Uygulamadan teorik olarak aldığımız eğitim ve test sonuçları 50 epoch değeri için %98,9 bandına oturduğu gözlemlenmiştir. Bu epoch sayısı artırıldığında oran miktarı ortalama %99 bandında kalacağı için eğitim ve test aşaması bu şekilde sonuçlandırılmıştır. Teorik olarak sonuçların başarı oranı yüksek çıkması doğru algoritmalar doğru parametreler ile doğru yolda olduğumuzu gösterse de bu çalışmayı gerçekleme yaparak sonuçların doğru olup olmadığını test etmek için çalışmamıza bizim oluşturduğumuz veri tabanını da eklenilerek uygulamanın gerçek hayatta uygulanabilirliği test edilmiştir. Bu aşamada kullanılan veri setleri eğitim için 10000 adet, test için 1000 adet ve gerçekleme için 500 adet 600x450 piksel çözünürlüğünde birbirinden tamamen farklı fotoğraflar kullanılarak gerçekleştirilmiştir. Veri seti hazırlığındaki klasörleme, etiketleme, veri seti artırma gibi tüm aşamalar ve eğitim-test esnasında kullanılan algoritmalar, sistemler, donanımlar ile ilgili daha detaylı bilgiye aşağıda bahsedilen model eğitim bölümünde yer verilmiştir. Çalışmada python programlama dili, CNN algoritması, tensorflow ve keras kullanılmıştır.

Literatürde yapılan çalışmalar incelendiğinde bu tarz araç marka model tanıma işlemleri için bir fotoğraf üzerinde tek tek araç görüntüsü etiketlemesi yapılarak veri seti oluşturulması ve bu veri setini daha önceden eğitilmiş Resnet50, Resnet101 ve türevleri gibi hali hazırda bulunan dosyalar kullanılarak test edildiği gözlemlenmiştir. Burada en temel konu hali hazırda eğitilmiş örnekler Resnet50, Resnet101 vb. gibi eğitim sonuç dosyaları hariç, kendi oluşturacağımız veri seti için resimleri tek tek açıp yardımcı bir program sayesinde etiketleme işleminin çok uzun zaman alacağı olması da ayrıca bir iş yüküdür.

İKİNCİ BÖLÜM
MATERYAL VE YÖNTEMLER

Bu çalışma tasarlanırken derin öğrenme kütüphanelerine kolayca erişim sağlamak hem iş yükünü azaltması hem de karmaşık kod yığınından kaçınılması için Python programlama dili tercih edilmiştir. Python editörleri ve diğer yardımcı programların kurulumlarına aşağıdaki bölümlerde yer verilmiştir. Sistem tasarlanma aşamasında derin öğrenme kullanılabilmesi için en önemli etken veri tabanının büyüklüğüdür. Ne kadar çok veri o kadar doğru sonuç alınması demektir. Bu sebepten dolayı veri seti sıfırdan oluşturulmuş ve yeterli sayıya ulaşabilmek için bazı yöntemler sayesinde veri seti büyüklüğü artırılmıştır. Veri artırma ile ilgili adımlara da aşağıdaki bölümlerde yer verilmiştir. Sistemin temel çalışma mantığı, girdi olarak verilen araç görüntüsü üzerinde daha önce den eğitilmiş veri seti sonuçlarını baz alarak bir model bilgisi çıktısı beklenir. Model çıktı bilgisi alınan araca ait sıradaki işlem aracın renk bilgisidir. Daha sonra son aşama olan plaka bilgisi de alınan araca ait bilgiler sanal olarak oluşturulan veri tabanındaki bilgiler ile karşılaştırılıp sonucu ekrana yazdırılır. Araca ait marka/model bilgisi, renk bilgisi veya plaka bilgisi gibi bilgilerden herhangi birisi tespit edilemediği taktirde işlem sonlandırılır. Bu bilgilerden herhangi birinin yanlış tespit edilmesi durumda ise veri tabanındaki bilgiler ile eşleşme sağlanamayacağı için eşleşmenin başarısız olduğu bilgisi verilerek işlem sonlandırılır. Bu işlemler ile ilgili akış diyagramı Şekil 2.1'de gösterildiği gibidir.



Şekil 2.1: İşlem akış diyagramı

2.1. Kullanılan Yazılım Platformları

Anaconda, veri bilimi üzerine bilimsel uygulamalar için python programlama dili kullanılarak çalışmalar yapmak üzere birçok paket programı tümleşik olarak içeriğinde barındıran ücretsiz bir python dağıtımıdır. Kurulum esnasında birçok

yapay zeka ve veri bilimi için kullanılan kütüphanelerin yanı sıra bazı Python IDE' leri de kurulu halde gelecektir. Kurulum için öncelikle işletim sisteminizle uyumlu olan versiyonu www.anaconda.com/ products/individual adresinden indirmeniz gereklidir. Bu yöntem haricinde komut istemi (CMD) ya da Linux ortamında Shell komutu yöneticisinden de kurulum gerçekleştirilebilir. Burada normal şekilde Windows işletim sistemi üzerine kurulum tercih edilmiştir. Kurulum aşamasının adımları esnasında indirilen exe çalıştırıldığında ilk adım next hemen peşinden lisans sözleşmesini kabul et dedikten sonra bize tüm kullanıcılar ya da sadece bu kullanıcı gibi bir seçenek soracaktır. Tercih edilen durum seçilerek next ile ilerlemeye devam edilir. Gelen üç adımda next deyip ilerliyoruz ve finish butonun gördüğümüzde kurulum tamamlanmış olacaktır. Anaconda kurulumu tamamlandıktan sonra ikonu çift tıklanarak açılır.

Bu Python IDE' leri içinde ihtiyaç duyduğumuz ve kullanacağımız Spyder ve Jupyter Notebook görüldüğü üzere kurulu halde karşımıza gelir.

Spyder Python; Spyder, python programlama dilinde gelişmiş düzenleme, etkileşimli test, hata ayıklama ve iç gözlem özellikleriyle Python dili için güçlü açık kaynaklı bir etkileşimli geliştirme ortamıdır. Ek olarak, Spyder, IPython ve NumPy, SciPy veya matplotlib gibi popüler Python kitaplıklarının desteği sayesinde sayısal bir hesaplama ortamıdır. İlk olarak 2009 yılında Pierre Raybaut tarafından oluşturulan ve geliştirilen Spyder, 2012'den beri bilimsel Python geliştiricileri ve topluluk tarafından sürdürülmekte ve sürekli olarak geliştirilmektedir.

Jupyter Notebook; Jupyter Notebook tıpkı spyder gibi python dilinde yazılan kodu çalıştırmaya ve test etmeye yarayan ücretsiz açık kaynaklı bir IDE' dir. Spyder' dan en önemli farkı ise canlı kod dedikleri yazılan kod satır bloğunu anında çalıştırıp sonucu ekranda görebilmemizdir.

2.2. Görüntü İşleme Kütüphaneleri

2.2.1. Tensorflow

Bu tez kapsamında Google geliştiricileri tarafından 2015 yılında ortaya çıkarılan açık kaynak kodlu bir derin öğrenme kütüphanesi kullanılmıştır. Geliştiricilerin makine öğrenmesi veya derin öğrenme üzerine modellerini tasarlarlamada çok büyük kolaylıklar sağlamıştır. Tensorflow, görüntü tanıma, doğal dil işleme, diferansiyel

denklemler, metin tamamlama vb. sınıflandırma uygulamaları için modelleri eğitmeye veya çalıştırmaya yarayan bir kütüphanedir. Bir veya daha fazla CPU ya da GPU' u üzerinde çalışmamızı sağlayan yapısı sayesinde karmaşık hesaplamaları veya sınıflandırma işlemlerini kolaylıkla gerçekleştirebilmemize olanak sağlar. Python kullanılarak geliştirilen bu framework günümüzde python haricinde Javascripti, R, Swift gibi birçok programlama dilini desteklemektedir (Z. Tan, 2019) (Abdulkadir Şeker, 2017).

Tensorflow, uygulama geliştiricileri tarafından oluşturulan veri akışlarının grafiğini hesaplayan, grafikteki her düğüm matematiksel bir işlem ifade ederken aynı zamanda düğümlerin arasındaki bağlantı çok boyutlu veri dizisi olarak ifade edilirler. Bu diziler bir başka deyişle tensörler olarak temsil edilirler (Goyal, n.d.).

Tensorflow bahsedildiği üzere hem CPU hemde GPU üzerinde çalıştırılabilir. Fakat buradaki temel sorun GPU desteğini şu anda sadece NVIDIA markasına veriyor olmasıdır. Ancak GPU üzerinden çalıştırılması bununla da tam olarak mümkün değil. Çünkü bu işlem yeni nesil bir ekran kartlarını destekleyen büyük iş yükü olan bir framework' dür. Bunun için NVIDIA' nın sitesinde CUDA Capacity adı altında GPU modelleri belirtilerek karşılığında hesaplama güç puanlarının listesi verilmiştir. GPU desteğinizin var olup olmadığını kurulum yapmadan önce kontrol etmeniz gerekmektedir. Son güncelleme ile birlikte bu kapasite hesaplama puanı 3.5 olarak güncellenmiştir. Bu hesaplama puanına karşılık gelen veya üst kapasite hesaplama puanına sahip olmayan bir GPU için kurulum yapılsa bile kodunuzun derlenmesi esnasında derleyici tarafından yetersiz hesaplama kapasitesi şeklinde hata mesajı verilerek CPU' nun çalıştırılmasına yönlendirilirsiniz. Fakat destekleyici bir GPU' ya sahipseniz kesinlikle GPU kurulumu ve üzerinde çalışmanız tavsiye edilir. Aralarındaki fark özetle CPU' nun yaklaşık 10 katı kadar daha hızlı olması olarak değerlendirilebilir.

CPU sürümü kurulumu için pip3 install tensorflow

GPU sürümü kurulumu için pip3 install tensorflow-gpu

Fakat GPU kurulumu için sadece tensorflow kurulması yeterli değildir. Bunun için ayrıca NVIDIA GPU Driver güncellemesi ve CUDA gibi birkaç gerekli eklentininde kurulması gereklidir. Bu konu için daha ayrıntılı bilgi tensorflow-gpu kurulumu adı altında araştırarak yapmanız mümkündür.

2.2.2. Keras

Keras, Python' da yazılmış açık kaynaklı bir yapay sinir ağı kütüphanesidir. 2017 yılı itibari ile Google' ın Tensorflow ekibi tarafından tensorflow çekirdek kütüphanesinde Keras' ı desteklemeye karar verildi. Keras katmanlar, aktivasyon fonksiyonları gibi parametreleri python dilini kullanarak yeni modeller oluşturmaya yarayan bir kütüphanedir. Kullanımı kolay basit ve sadedir. Şu an aktif kullanılan sürümü arka planında Tensorflow backend üzerine inşa edilmiştir. En temel mantıkla tüm yapı katmanlara dayanır. Her yapılacak işlem katmanlar arasında gerçekleşir. İlk katmanda veri girişi olurken son katmanda veri çıkışı sağlanır. Giriş ve Çıkış katmanı dahil bu yapıya model adı verilir. Modelin eğitimi de bu aralıkta gerçekleşir.

2.2.3. Numpy

Numpy, Python' da bilimsel hesaplamalarda kullanılan temel bir matematik kütüphanesidir. Temelinde çok boyutlu numpy dizileri bulunan bu yapı bir dizi içinde çeşitli diziler veya matrisler barındıran, matematiksel işlemler veya mantıksal işlemler kullanımı esnasında ihtiyaç duyulur. Kurulum için; pip3 install numpy

2.2.4. PIL(Pillow) – Python Image Library

Python' da bir görüntü üzerinde herhangi bir işlem yapabilmek için kullanılan bir grafik işleme kütüphanesidir. İçerisinde birçok fonksiyon barındıran bu kütüphane çizim, düzenleme veya filtreleme gibi işlemleri hazır fonksiyonları sayesinde kullanıcıya hızlı bir görüntü işleme imkânı verir. Kurulum için; pip3 install pillow

2.2.5. İmutils

Python' da bir görüntü üzerinde çevirme, döndürme, yeniden boyutlandırma, kontur çıkarma, kenar algılama vb. gibi morfolojik işlemleri yapabilmemizi sağlayan bir kütüphanedir. Kurulum için; pip3 install imutils

2.2.6. PyTesseract

Python-tesseract, python' da optik karakter tanıma aracıdır (Ray, 2011). Pillow tarafından desteklenen görüntü uzantılarının tamamını destekler. Örneğin jpeg, png, tiff, bmp, gif gibi uzantılı görüntüleri okuyup karakter olarak çıktısını sunan bir kütüphanedir. Kurulum için; pip3 install pytesseract

2.2.7. Skimage

Python' da bir görüntü üzerinde segmentasyon, geometrik dönüşümler, renk alanı manipülasyonu, analiz, filtreleme, morfoloji, özellik algılama gibi işlemleri gerçekleştirebilmek için kullanılır.

2.2.8. OpenCV

Open Source Computer Vision yani açık kaynak kodlu görüntü işleme kütüphanesidir. İlk olarak 1999 yılında intel firması geliştirilen bu kütüphane AMD, Google ve Nvidia gibi şirketlerin çalışmaları ile beraber gelişmesine katkı sağlanmıştır. Temelinde C programlama diline uyumlu olarak geliştirilen bu kütüphane, daha sonra kullanılan tüm programlama dillerine entegre edilebilecek şekilde geliştirilmiş ve geliştirilmeye devam edilmektedir.

Temel olarak OpenCV kütüphanesi içerisinde görüntü işlemeye yarayan ve makine öğrenmesi gibi çalışmalara katkı sağlayacak 2500' den fazla algoritması olduğu biliniyor. Bu algoritmalar sayesinde yüz tanıma, nesneleri sınıflandırma, insan hareketlerini tespit etme, plaka tanıma, 3B resimler üzerinde işlem yapabilme, görüntü karşılaştırma, OCR (Optical Character Recognition) Optik Karakter Tanıma gibi birçok işlemin yapılabilmesine imkân veren bir image processing(görüntü işleme) kütüphanesidir. Beş temel bileşenden oluşmaktadır. Bunlar;

- **CV (Computer Vision-Bilgisayarla Görme) Bileşeni:** Bu bileşen görüntü işleme fonksiyonlarını ve algoritmalarını içerir.
- **MLL (Machine Learning Library) Bileşeni:** Bu bileşen adında anlaşılacağı üzere makine öğrenimi için gerekli olan fonksiyonları ve algoritmaları içerir.
- **HighgGUI Bileşeni:** Bu bileşen resim veya videoların kaydedilip silinmesi işlemlerini yerine getiren fonksiyonları içerir.
- **CXCore Bileşeni:** Resim üzerinde çizim yapmamızı sağlayan ve XML desteği veren bileşendir.
- **CvAux Bileşeni:** Şablon eşleştirme, yüz tanıma veya dudak hareketlerini izleme gibi algoritmalar içeren bir bileşendir.

Kurulum için; pip3 install opencv-python

2.3. Veri Tabanı İşlemleri

Veritabanı, herhangi bir depolama ünitesi içerisinde yer alan bilgi veya verilerin düzenli halde tutan veri bütünüdür. Programlamanın yanı sıra verilerin saklanıldığı kısma da erişim, güncelleme, ekleme, silme veya sorgulama gibi işlem için kullanılan bu yapı üniversite, okul, hastahane, seyahat, şirket vb. kuruluşların çalışmasında hatta kendi aralarında entegre olup ortak bir yönetim ağında çalıştığında gerekli olan en temel yapıdır. Aynı zamanda büyük bilgi verilerinin depolandığı, gerekli olduğunda erişime olanak sağlar.

Bu tez kapsamında DB Browser for SQLite(DB4S) programı kullanılmıştır. İşletim sisteminize uygun versiyonu https://sqlitebrowser.org/dl/ sitesinden ücretsiz bir şekilde indirip kurabilirsiniz. Bu programın kullanılmasındaki en temel amaç programın sade ve basit bir kullanım yapısına sahip olması ve kurulum esnasında herhangi bir server ile ilişkilendirme ve localhost' dan çağırma vs. durumların olmamasıdır. Programın kurulumundan kullanımına kadar basit bir yapı sunuyor olmasıdır.

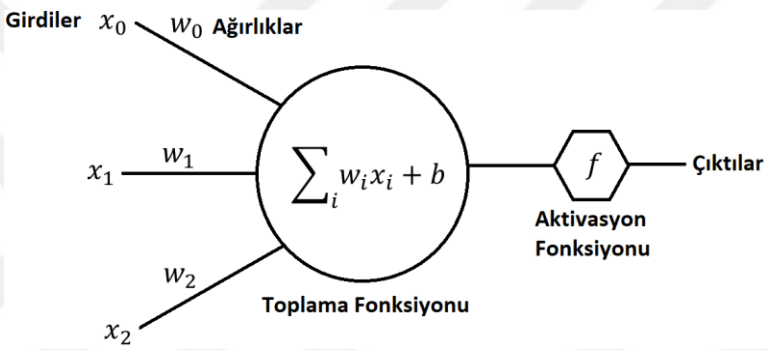
DB4S, veritabanları oluşturmak, aramak ve düzenlemek isteyen kullanıcılar ve geliştiriciler içindir. DB4S kullanım arayüzünün sadeliği sayesinde karmaşık SQL komutlarının öğrenilmesine ihtiyaç duymaz. Kullanımı oldukça basit olan bu yapı arayüzü sayesinde aşağıda bahsedilen adımları kolay ve hızlı bir şekilde gerçekleştirebilir;

- Veritabanı dosyaları oluşturun ve sıkıştırın
- Tablo oluşturun, tanımlayın, değiştirin ve silin
- Dizinler oluşturun, tanımlayın ve silin
- Kayıtlara göz atın, düzenleyin, ekleyin ve silin
- Kayıtları ara
- Kayıtları metin olarak içe ve dışa aktar
- Tabloları CSV dosyalarından / dosyalarından içe ve dışa aktarın
- Veritabanlarını SQL döküm dosyalarından içe ve dışa aktarın
- SQL sorguları yayınlayın ve sonuçları inceleyin
- Uygulama tarafından verilen tüm SQL komutlarının günlüğünü inceleyin
- Tablo veya sorgu verilerine dayalı basit grafikler çizin

Bu çalışma kapsamında araç görüntüsü alındıktan sonra marka/model bilgisi, plaka bilgisi ve renk bilgisi tespit edilir. Tespit edilen bilgiler doğrultusunda veri tabanı kontrol edilir. Marka/model, plaka ve renk bilgisi karşılaştırılan araç kayıtlarda doğru bir şekilde yer alıyorsa aracın güvenli olduğu ya da bilgilerde eşleşme konusunda olumsuz bir durum olduğunda kullanıcıya eşleşmenin başarısız olduğu bilgisini vermek amacıyla kullanılmıştır.

2.4. Bazı Aktivasyon Fonksiyonlarının İrdelenmesi

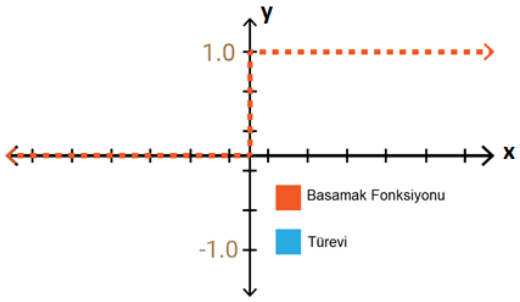
Yapay sinir ağlarında doğrusal olmayan özellikleri tanıtabilmek için aktivasyon fonksiyonları kullanılır. Basit bir yapay sinir ağı içerisinde girdiler için x, ağırlıklar için w ve çıkışa eklenerek aktarılan f(x) aktivasyon adımları ile Şekil 2.2' deki gibi gösterilir.



Şekil 2.4.1: Aktivasyon fonksiyonunun yapay sinir ağına uygulanması

Bu çıkış sonuç çıkışı ya da bir başka katman için giriş değeri olur. Aktivasyon fonksiyonu olmayan bir sinir ağı doğrusal fonksiyon olarak hareket eder. Dolayısıyla sinir ağı tek dereceli polinomlardan oluşan sınırlı öğrenme gücüne sahip kalırlar. Burada istenilen şeyin yapay sinir ağında oluşacak doğrusal olmayan durumlarında öğrenilmesi temel hedeftir. Bunun sebebi sinir ağımız içerisinde öğrenmesi için görüntü, video, yazı ve ses gibi gerçek dünya bilgilerinin verilecek olmasıdır. Bu sayede sinir ağımız daha hızlı ve doğru öğrenme gerçekleştirir. Ağırlıkların hesaplanabilmesi için hatanın geriye yayılımı algoritması kullanılmaktadır. Burada en temel strateji hata oranını minimize etmektir. Bu işlem Şekil 2.2' de gösterildiği gibi girişler ile ağırlıkları çarp, aktivasyon fonksiyonunu sağa veya sola ötelenmesini sağlayan sapma/eğilim (bias) değeri ile topla ve aktivasyonu uygula şeklinde bir adımdır (Lau & Lim, 2019), (H. H. Tan & Lim, 2019) .

2.4.1. Basamak (Step) Fonksiyonu



Şekil 2.4.1.1: Basamak Fonksiyonu

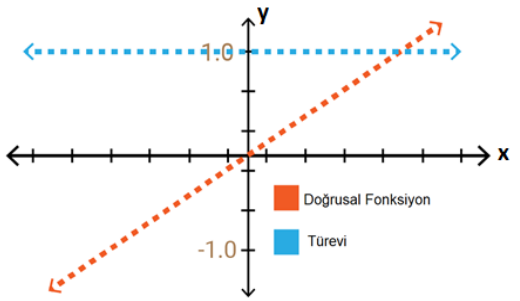
(Yi, 2018)

İkili sınıflandırıcı olarak kullanılan bu fonksiyon ikili değer alır. Bu nedenle çıkış katmanında kullanılması tercih edilir. Gizli katmanlarda türevi öğrenme değeri oluşmadığı için kullanılmaz (Sharma V. Avinash, 2017).

$$f(x) = \begin{cases} 0 \; için \; x \; < \; 0 \\ 1 \; için \; x \; \geq \; 0 \end{cases} \quad (2.4.1.1)$$

$$f^{l}(x) = \begin{cases} 0 \; için \; x \; \neq \; 0 \\ ? \; için \; x \; = \; 0 \end{cases} \quad (2.4.1.2)$$

2.4.2. Doğrusal (Linear) Fonksiyonu



Şekil 2.4.2.1: Doğrusal Fonksiyonu

(Yi, 2018)

Basamak fonksiyonun aksine türev öğrenmesi vardır. Bir dizi aktivasyon değeri üretirken aynı zamanda bu ürettiği değerler basamak fonksiyonundaki gibi ikili

değerlerden oluşmaz. Birçok nöronun birbirine bağlanmasını sağlar. Fakat bu fonksiyonunda bir handikabı mevcuttur. Bu handikap türevin sabit olmasıdır. Öğrenme işlemi nöronlar ile gerçekleştirilir. Öğrenme esnasında türev alınır. (2.4.2.3) için x' e göre türev alındığında a sonucuna varılır. Burada x değerinin bir anlamı kalmaz. Yani türev sürekli sabit bir değer çıkıyor ve öğrenme işlemi gerçekleşmiyor.

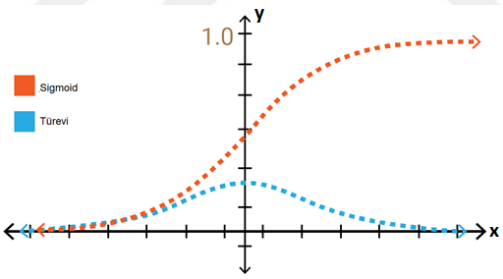
$$f(x) = x \quad (2.4.2.1)$$

$$f^{l}(x) = x \quad (2.4.2.2)$$

$$T = a.x \quad (2.4.2.3)$$

Diğer bir sorunu ise bu fonksiyonun, her katmanda doğrusal fonksiyon kullanıldığı zaman giriş ve çıkış katmanı arasında her zaman aynı doğrusal sonuca ulaşılır. Bu fonksiyonun kullanımı ara katmanların işlevsiz kalması anlamına gelmektedir (Sharma V. Avinash, 2017).

2.4.3. Sigmoid Fonksiyonu



Şekil 2.4.3.1: Sigmoid Fonksiyonu

(Yi, 2018)

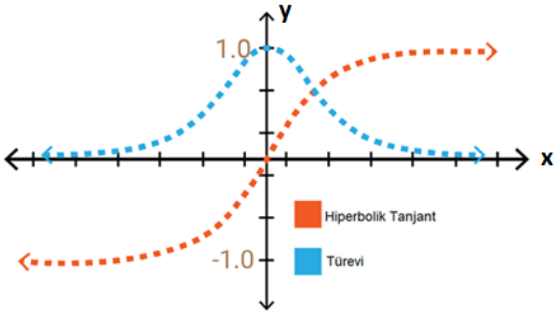
Türevlenebilen bu fonksiyon için grafiğe bakıldığında x, -2 ile +2 değer aralığında iken y ekseninde değerlerin hızlı bir şekilde değişiklik gösterdiği gözlemlenir. x ekseninde yapılabilecek küçük bir değişimin y ekseninde büyük değişikliklere sebep olacağı görülmektedir. Bu iyi bir sınıflandırıcı olarak kullanılabilir anlamına gelmektedir. Diğer avantajlı durumu ise tıpkı doğrusal fonksiyonda olduğu gibi (-sonsuz, +sonsuz) ile karşılaşıldığında her zaman için (0,1) aralığında değer üretmesidir. Bu bölgelerde türev değerleri çok küçük olur ve 0' a yakınsar. Bu olaya

gradyanların ölmesi ya da kaybolması denir. Öğrenme olayı minimum düzeyde gerçekleşir. Değer 0 olursa gerçekleşmez. Exp() hesaplamasının yavaş olması doğal olarak yavaş öğrenme olayının gerçekleşmesine sebep olmaktadır (Sharma V. Avinash, 2017), (Sharma, 2019), (Chung et al., 2016).

$$f(x) = \sigma(x) = \frac{1}{1 + e^{-x}} \quad (2.4.3.1)$$

$$f^{l}(x) = f(x)\,(1 - f(x)) \quad (2.4.3.2)$$

2.4.4. Tanh Fonksiyonu



Şekil 2.4.4.1: Tanh Fonksiyonu

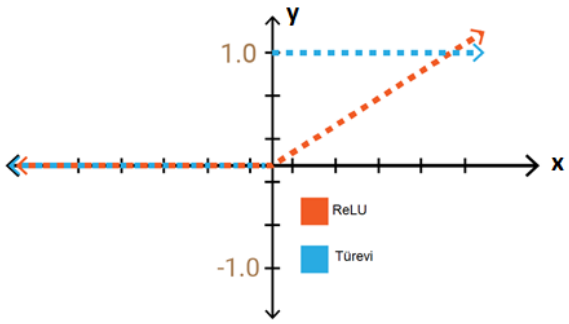
(Yi, 2018)

Grafikten de anlaşılacağı üzere Sigmoid fonksiyonuna çok benzeyen bir yapısı vardır. Fakat en temel farkı fonksiyon aralığının bu fonksiyon için (-1, +1) olarak tanımlanmasıdır. Sigmoid fonksiyonuna göre türevin daha dik olması yani daha çok değer alabilmesidir. Bu demek oluyor ki hızlı öğrenme ve sınıflandırma işlemi için daha geniş değer aralığına sahip demektir. Bu öğrenmenin verimli olduğu anlamına gelse de gradyanların ölmesi probleminin devam ettiği görülmektedir (Sharma V. Avinash, 2017), (Chung et al., 2016).

$$f(x) = tanh(x) = \frac{e^{x} - e^{-x}}{e^{x} + e^{-x}} \quad (2.4.4.1)$$

$$f^{l}(x) = 1 - f(x)^{2} \quad (2.4.4.2)$$

2.4.5. ReLU Fonksiyonu



Şekil 2.4.5.1: ReLU Fonksiyonu

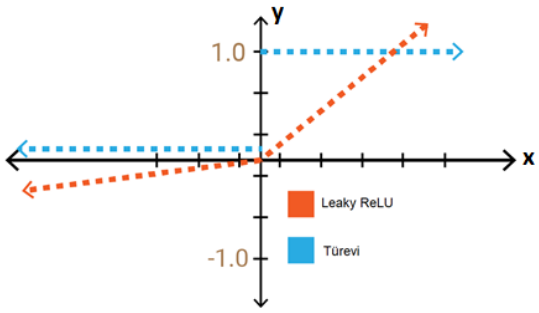
(Yi, 2018)

Grafiğe bakıldığında pozitif eksende doğrusal fonksiyon ile aynı olduğu görülmektedir. Fakat ReLU doğrusal değildir. ReLU[0, +∞) aralığında değer alır. Sigmoid ve hiperbolik tanjant fonksiyonlarında tüm nöronların aynı şekilde aktive oluyordu. Bu aktivasyon yoğun yani çok işlem gerektiriyor demektir. Fakat ağdaki bazı nöronların aktif olmasını yani aktivasyonun seyrek olması istenilir. Sebebi ise daha az işlem yükü ile daha verimli sonuçlar elde edebilmektir. ReLU tam olarak bu nokta da negatif eksendeki değerler için 0 değerini almasıdır. Bu öğrenme işleminin hızlı gerçekleşeceği anlamına gelirken bir handikabı ortaya çıkardığı görülmüştür. Fonksiyona hız kazandıran bu sıfır bölgesinde türevinin de sıfır olması demektir. Yani öğrenme olayının bu bölgelerde gerçekleşmediği anlamına gelir (Sharma V. Avinash, 2017), (Sharma, 2019), (Chung et al., 2016).

$$f(x) = \begin{cases} 0\ için\ x\ <\ 0 \\ x\ için\ x\ \geq\ 0 \end{cases} \tag{2.4.5.1}$$

$$f^{l}(x) = \begin{cases} 0\ için\ x\ <\ 0 \\ 1\ için\ x\ \geq\ 0 \end{cases} \tag{2.4.5.2}$$

2.4.6. Leakly (Sızıntı) ReLU Fonksiyonu



Şekil 2.4.6.1: Leakly ReLU Fonksiyonu

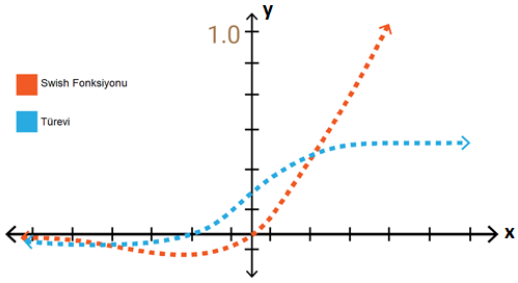
(Yi, 2018)

Grafikten de anlaşılacağı üzere ReLU ile çok benzer fakat negatif yönlerinde iyileştirme yapılmış bir fonksiyon görüntüsüne bürünmüştür. İyileştirme yapılan negatif bölgesi için bu sızıntı değeri 0,01 olarak verilir. Eğer sıfıra yakın farklı bir değer verilirse fonksiyonun adı rastgele Leaky ReLU olarak adlandırılır. Bu Leaky ReLU fonksiyonunun değer aralığı negatif yönde eksi sonsuza doğru gitmektedir. Bu değer 0' a yakın fakat 0 olmayan değer sayesinde ReLU fonksiyonunda ölen gradyanların yaşatılması yani öğrenmenin negatif bölgedeki değerler için de devam ettirilmesidir (Sharma V. Avinash, 2017), (Sharma, 2019).

$$f(x) = \begin{cases} 0.01x \; için \; x < 0 \\ x \qquad için \; x \geq 0 \end{cases} \tag{2.4.6.1}$$

$$f^{l}(x) = \begin{cases} 0.01 \; için \; x < 0 \\ 1 \quad için \; x \geq 0 \end{cases} \tag{2.4.6.2}$$

2.4.7. Swish Fonksiyonu



Şekil 2.4.7.1: Swish Fonksiyonu

(Yi, 2018)

ReLU fonksiyonuna göre en önemli farkı negatif bölgede değer alıyor olmasıdır. Fakat Leaky ReLU fonksiyonu da negatif bölgede değer alıyordu ancak aralarındaki en temel fark swish' in negatif bölgede almış olduğu değerler doğrusal değildir. Diğer tüm aktivasyon fonksiyonları monotondur. Swish fonksiyonunun çıktısı girdi arttığında bile düşebildiğidir. Eğer beta değeri 0 seçilir ise sonuç doğrusal olur ve ReLU fonksiyonuna benzer. Bu nedenle beta değeri 1 olarak seçilerek istenilen hassasiyet sağlanmış olur (Sharma V. Avinash, 2017).

$$f(x) = x\,.\,\sigma(x) \tag{2.4.7.1}$$

$$f^{l}(x) = f(x) + \sigma(x)(1 - f(x)) \tag{2.4.7.2}$$

2.4.8. Softmax Fonksiyonu

Sigmoid fonksiyonuna benzeyen bir yapıdadır. İkiden fazla sınıflandırma problemi olduğunda modelin çıkış katmanında kullanılmaktadır. (Nwankpa et al., 2018) 0-1 aralığında olasılık değeri üreterek girdinin belirli sınıfa ait olup olmadığının belirlenmesi sağlar. (Sharma V. Avinash, 2017), (Sharma, 2019).

$$f_i(\vec{x}) = \frac{e^{x_i}}{\sum_{j=1}^{J} e^{x_j}} \quad \text{için } i = 1, \ldots, J \tag{2.4.8.1}$$

$$\frac{\partial f_i(\vec{x})}{\partial x_j} = f_i(\vec{x})(\delta_{ij} - f_j(\vec{x})) \tag{2.4.8.2}$$

2.5. Google Colab (Google Colaboratory)

Temel olarak Jupyter notebook ortamını kullanarak browser üzerinden Google Bulut servislerine bağlanıp hazırlanmış olan kodları sanal bir bilgisayar üzerinde çok daha hızlı çalıştırma imkânı veren ücretsiz bir geliştirici ortamıdır (Carneiro et al., 2018). Hizmetin ücretsiz olmasının yanı sıra ücretli versiyonuda mevcuttur. Aralarındaki en temel fark ücretsiz hizmetin ortalama 12 saat çalıştırılabilmesi imkânı verilirken, ücretli hizmeti ortalama 24 saat gibi bir süre ile hizmet verebilmesidir.

Google Colab' ın GPU (Graphics Processing Unit) ve TPU (Tensor Processing Unit) olmak üzere iki hizmeti mevcuttur. Bu iki hizmetin ayrımını en basit şekilde ifade etmek gerekirse, görüntü işleme, eğitim ve test işlemleri için GPU, matematiksel matris hesaplama işlemleri için TPU kullanıldığı gözlemlenilmiştir. Bu genellemeyi gözlemlemek için uygulamamızda GPU ve TPU ayrı ayrı seçilerek çalıştırılmış 1

epoch süresinin GPU' nun TPU' ya göre daha hızlı olduğu görülmüştür. Dolayısıyla uygulamamızda Google Colab GPU hizmeti kullanılmıştır.

Colab, mevcut kullanılan bilgisayarların işlem gücünün yetersiz kaldığı durumlarda veya büyük veri setleri üzerinde yapılacak çalışmalarda uygulamanın çalışma süresinin günler hatta haftalar sürmesi beklenirken bu işlemleri Nvidia Tesla K80 GPU donanımı sayesinde çok hızlı bir şekilde gerçekleştirir.

Colab anlık olarak sanal bir makine hizmeti verebilmektedir. Bunun için bir Google Hesabı açmanız ve çalıştırmak istenilen kodları Google Drive bölümüne yüklemeniz yeterlidir. Aynı zamanda eğitim/test gibi işlemler üzerinde çalışmak istenildiğinde de aynı adımlar izlenir. Eğitim ve test için kullanılacak klasörleri Google Drive bölüme yüklenilmesi yeterlidir. Birkaç adımda ücretsiz bir şekilde uygulamanızı çalıştırır ve hizmetin ne kadar hızlı sonuçlar ürettiğini gözlemleyebilirsiniz. Nvidia Tesla K80 GPU donanımın özellikleri gelişen teknoloji ile değişiklik gösterebilir (NVIDIA, 2015). Üretici firma tarafından paylaşılan donanım özellikleri şu şekildedir;

- Çift GPU tasarımlı 4992 CUDA çekirdeği
- NVIDIA GPU Boost ile 2.91 Teraflopa kadar çift hassasiyet performansı
- NVIDIA GPU Boost ile 8.73 Teraflopa kadar tek hassasiyet performansı
- 24 GB GDDR5 bellek (Ücretsiz versiyonunda 12GB RAM Desteği)
- 480 GB/s toplam bellek bant genişliği (Ücretsiz versiyonda ortalama 60GB)
- Veri güvenilirliği için ECC koruma
- Veri merkezinde en yüksek verimlilik için sunucu optimizasyonu

2.6. Model Eğitim

2.6.1. Veri Seti Oluşturma

Veri seti oluşturulurken internet tarayıcısından elde edilen görüntüleri tek tek kaydedip belirli aşamalardan geçirilerek eğitim ve test adımlarına hazır hale getirilmiştir. Bu adımlar ise şu şekilde;

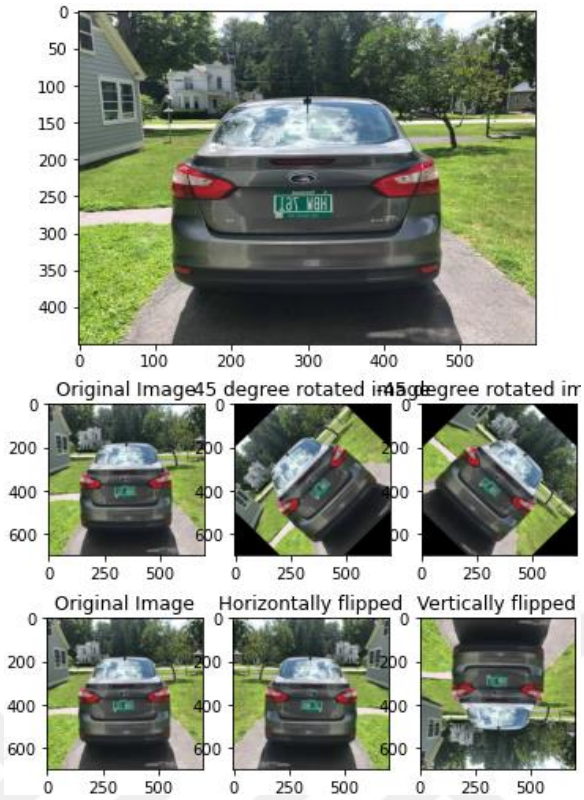
Şekil 2.6.1.1: Veri seti oluşturma adımları

- ***Etiketli Sınıf Klasörleri Oluştur,*** etiketleme olarak 2012_2014_Ford_Focus_F ile ön yüz, 2012_2014_Ford_Focus_R arka yüz, 2016_2019_Honda_Civic_F ön yüz ve 2016_2019_Honda_Civic_R arka yüz olacak şekilde etiketleme yapılarak 4 ayrı sınıf olarak değerlendirildi.

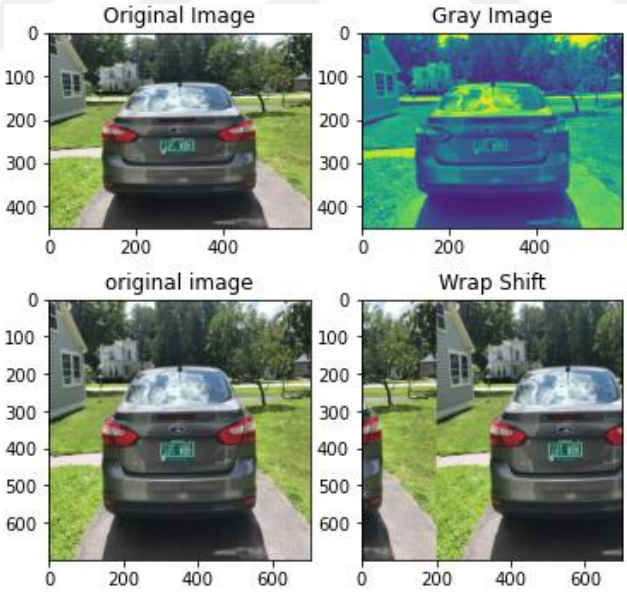
- ***Resim indir,*** free olarak erişime açık https://cfl.craigslist.org(*Craigslist.Org*, n.d.) ile https://www.autoscout24.com.tr(*Autoscout24*, n.d.) web siteleri içerisinde bulunan belirlediğimiz 2 marka/modele ait ön ve arka yüzden çekilmiş 600x450 piksel ölçülerinde yaklaşık 2500 adet araç görüntüsü elde edilmiştir. Belirlenen marka ve modeller ise 2012_2014_Ford_Focus ve 2016_2019_Honda_Civic' dir.

- ***İlgili klasöre aktar,*** elde edilen bu görüntüler klasörlere ayrılırken araçlar ön ve arka yüz şeklinde ayrıldı.

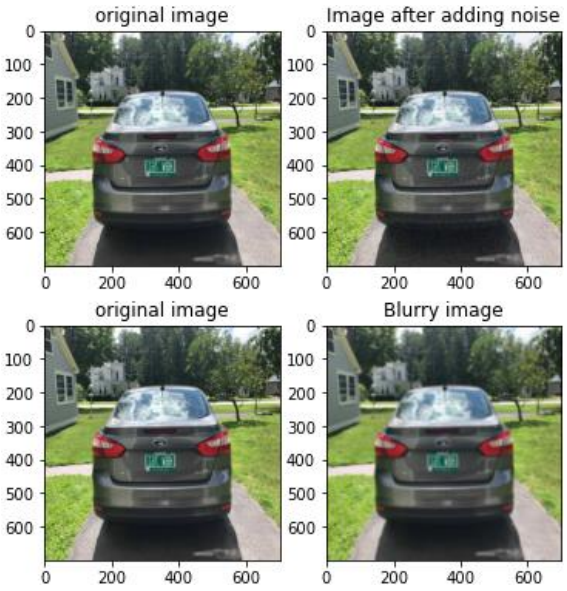
- ***Resim Sayısını Artırma İşlemleri,*** bu işlem literatürde data augmentation olarak geçen Türkçesi veri artırma olarak adlandırdığımız morfolojik işlemlerdir. Mevcutta bulunan 2500 görüntüyü döndürme, öteleme, kırpma, soldurma, ölçeklendirme, piksel ekleme gibi işlemler uygulanarak yeni görüntülerin elde edilmesidir. Bu şekilde oluşturulan yeni sentetik verilerle veri seti daha büyük bir eğitim seti haline dönüştürüldü. 1 resimden ortalama 40' a yakın rastgele yeni görüntüler elde edilmiş olup yaklaşık 108000 adet görüntü oluşturulmuştur. Bu işlemler sayesinde modelin ezberlemesinin de önüne geçilerek daha iyi sonuçlar elde edilmiştir (Mash et al., 2016).



Şekil 2.6.1.2: Veri Artırma_1



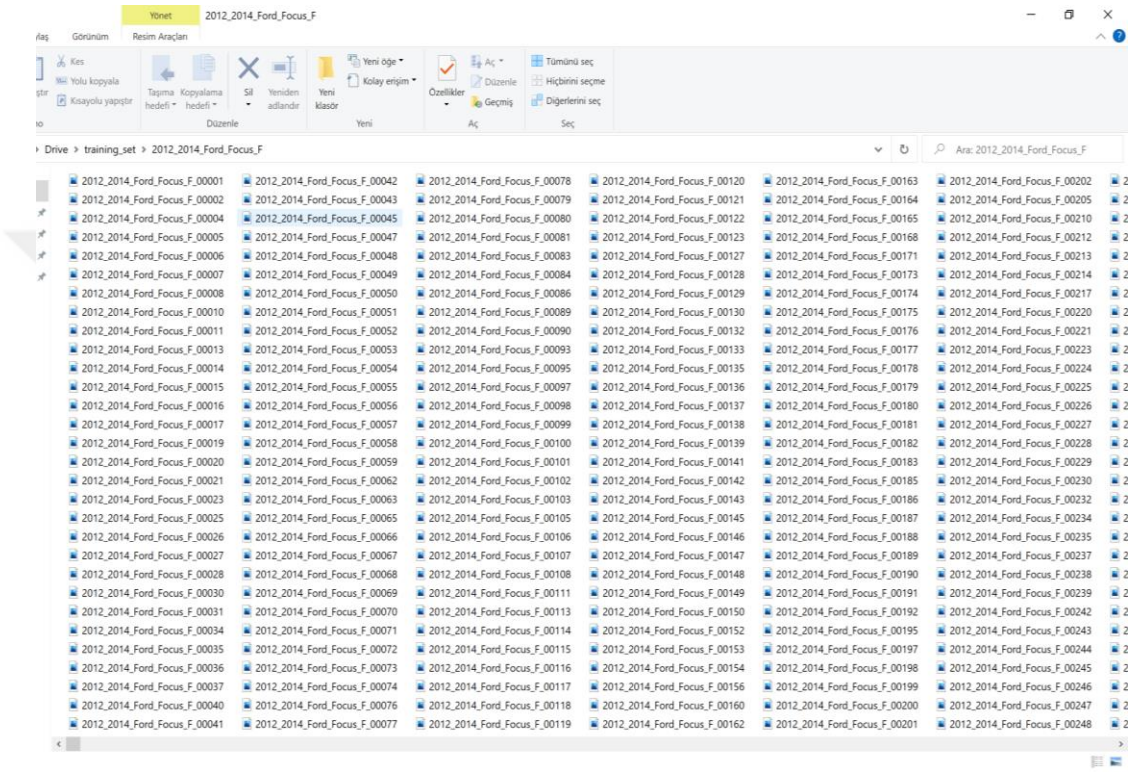
Şekil 2.6.1.3: Veri Artırma_2



Şekil 2.6.1.4: Veri Artırma_3

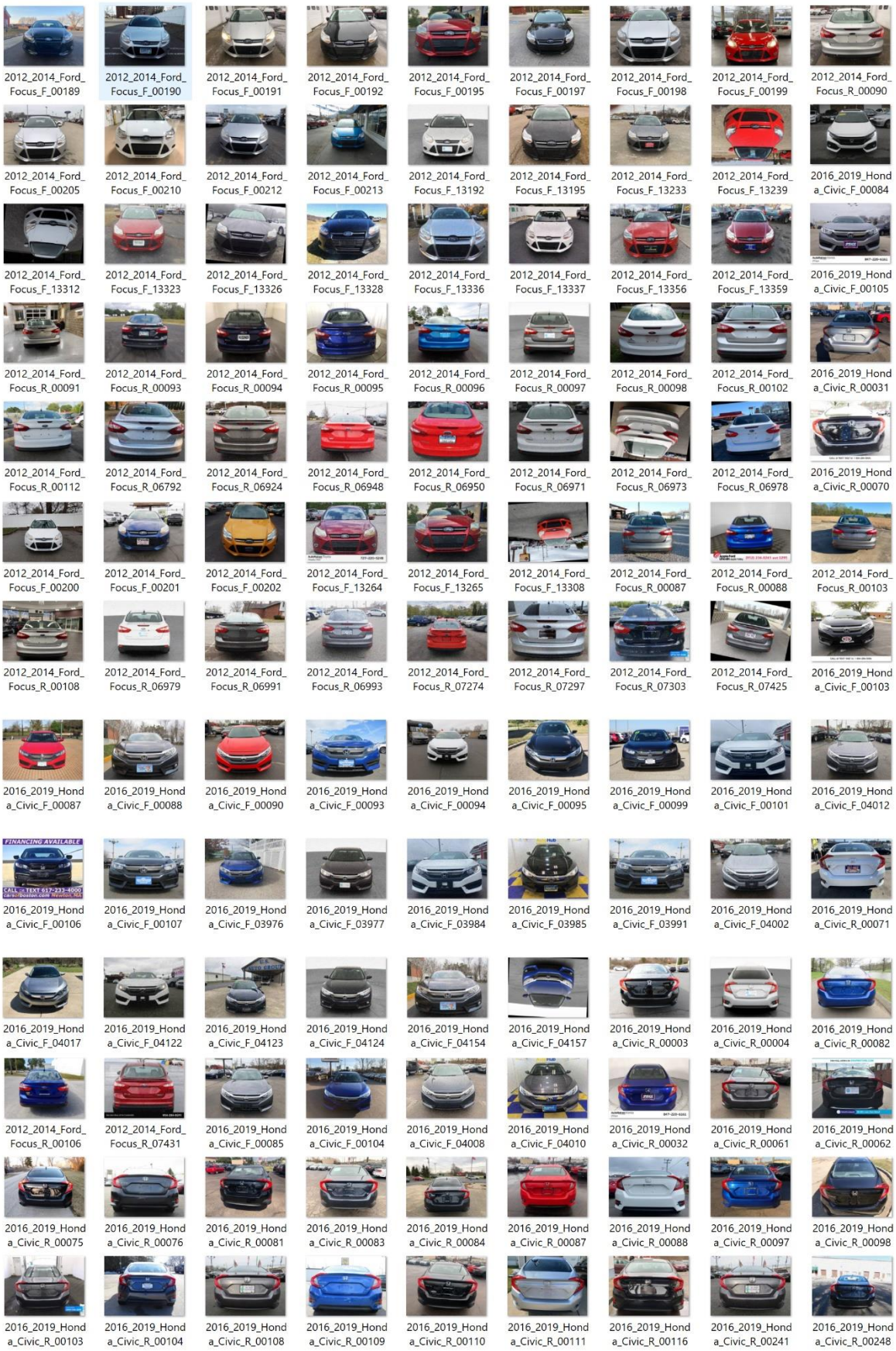
Şekil 2.6.1.5: Artırılmış Veriler Toplamı

- ***Benzer olan resimleri temizleme,*** oluşturulan sentetik veriler rastgele olarak oluşturulduğu için tamimiyle birbiriyle aynı olan resimler de üretilmiştir. Dolayısıyla modelin ezber yapmaması için birbirinden farklı resimler ile eğitilmesi gerekir. Yardımcı program sayesinde resimler tek tek karşılaştırılıp aynı görüntüler (boyut, ölçek vb.) özellikleri kıyaslanarak veri seti içerisinden temizlenmiştir.
- ***Resim adlarını etiketleme,*** her biri birinden farklı tüm veri seti klasör adıyla uyumlu olacak şekilde isim verilerek etiketlendirildi.



Şekil 2.6.1.6: Artırılmış Verilerin Etiketlenmesi

Veri seti Honda Civic ve Ford Focus marka/model araçlarının ön ve arka yüz görüntülerinden oluşmak üzere 4 ayrı sınıf olarak değerlendirilip 10000 adet görüntü ile eğitim, 1000 adet görüntü ile test edilmiştir. Şekil 2.6.1.7’ de gösterildiği gibi veri seti oluşturulmuştur.



Şekil 2.6.1.7: Veri Seti İçerisinden

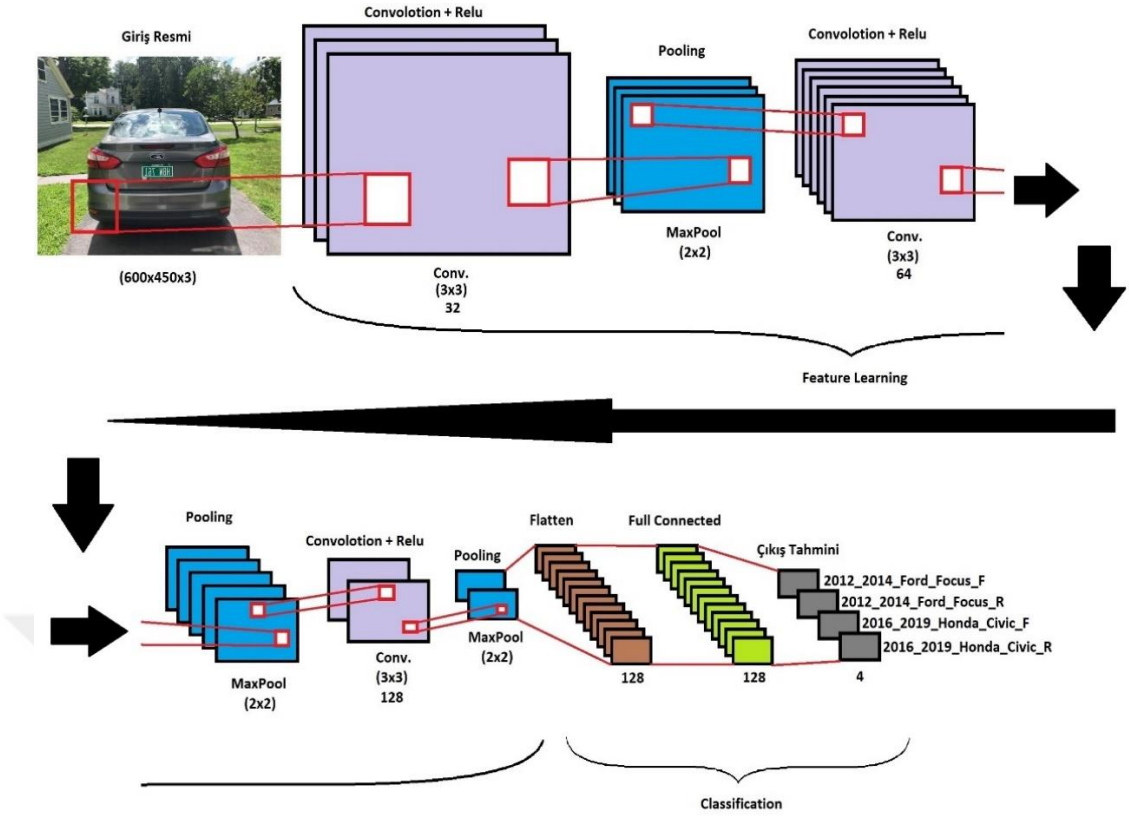
Fakat derin öğrenme için gereken büyük veri seti sayısına ulaşıp daha iyi sonuçlar alabilmek için indirilen veri setine veri artırma yöntemleri kullanılarak ortalama 40 katı kadar artırma işlemi ile yaklaşık 108000 adet resim oluşturuldu. Fakat donanım yetersizliği sebebiyle Tablo 1' de gösterildiği üzere veri seti sayısı 10000 adet eğitim, 1000 adet test, 100 adet gerçekleme(doğrulama) sayısı olmak üzere 4 adet sınıf olacak şekilde gerçekleştirildi. Eğitim ve Test için kullanılan araç görüntüleri plakasız şekilde, gerçekleme de kullanılan resimler plakalı şekilde kullanılmıştır.

Tablo 2.6.1.1: Eğitim - Test - Gerçekleme

Marka/Model	Eğitim	Test	Gerçekleme
2012_2014_Ford_Focus_F	2500	250	25
2012_2014_Ford_Focus_R	2500	250	25
2016_2019_Honda_Civic_F	2500	250	25
2016_2019_Honda_Civic_R	2500	250	25

2.6.2. Modelin Eğitilmesi

Modelin eğitimi için Konvolüsyonel (Evrişimli) Sinir Ağları (Convolutional Neural Networks) (CNN) mimarisi kullanılmıştır. Bu yöntem en sade biçimde anlatılmak istenirse bir görüntü üzerindeki çeşitli objeleri veya nesneleri birbirinden ayrıştırmamızı sağlayan derin öğrenme algoritmasıdır. Evrişimli Sinir Ağları, bir veya birden fazla evrişimli katmanlardan, altörnekleme katmanından ve standart çok katmanlı bir sinir ağı olmak üzere bir ya da biden fazla bağlı katmanlardan oluşur.

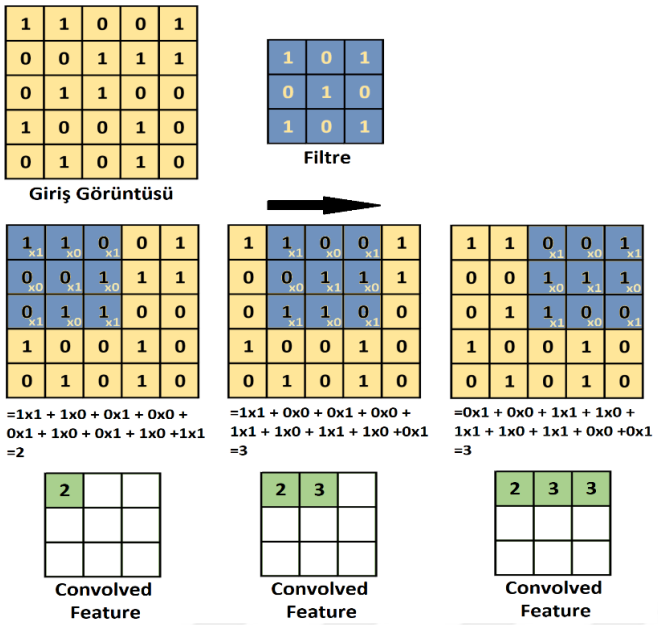


Şekil 2.6.2.1: Modelin Eğitim Adımları

Temel olarak CNN, sınıflandırma sonucunu standart sinir ağı kullanarak çözüme ulaştırır. Fakat bazı bilgileri ve özellikleri tespit edebilmek için diğer katmanlara da ihtiyaç duyar. Bu katmanları inceleyecek olursak;

2.6.2.1. Evrişim Katmanı (Convolutional Layer)

CNN' in ilk katmanıdır. Görüntü üzerine bazı filtreler uygulanarak düşük ve yüksek seviyeli özelliklerini görüntü üzerinde algılar. Bu katmanda resim girdi olarak gelir ve bir filtreden geçirilir. Filtreden geçirildikten sonra oluşan değerler sonucu bir öznitelik haritası oluşturur. Örneğin, bu aşama Şekil 2.6.2.1.1' de aşama aşama görülebilir.



Şekil 2.6.2.1.1: Öznitelik haritası oluşum adımları

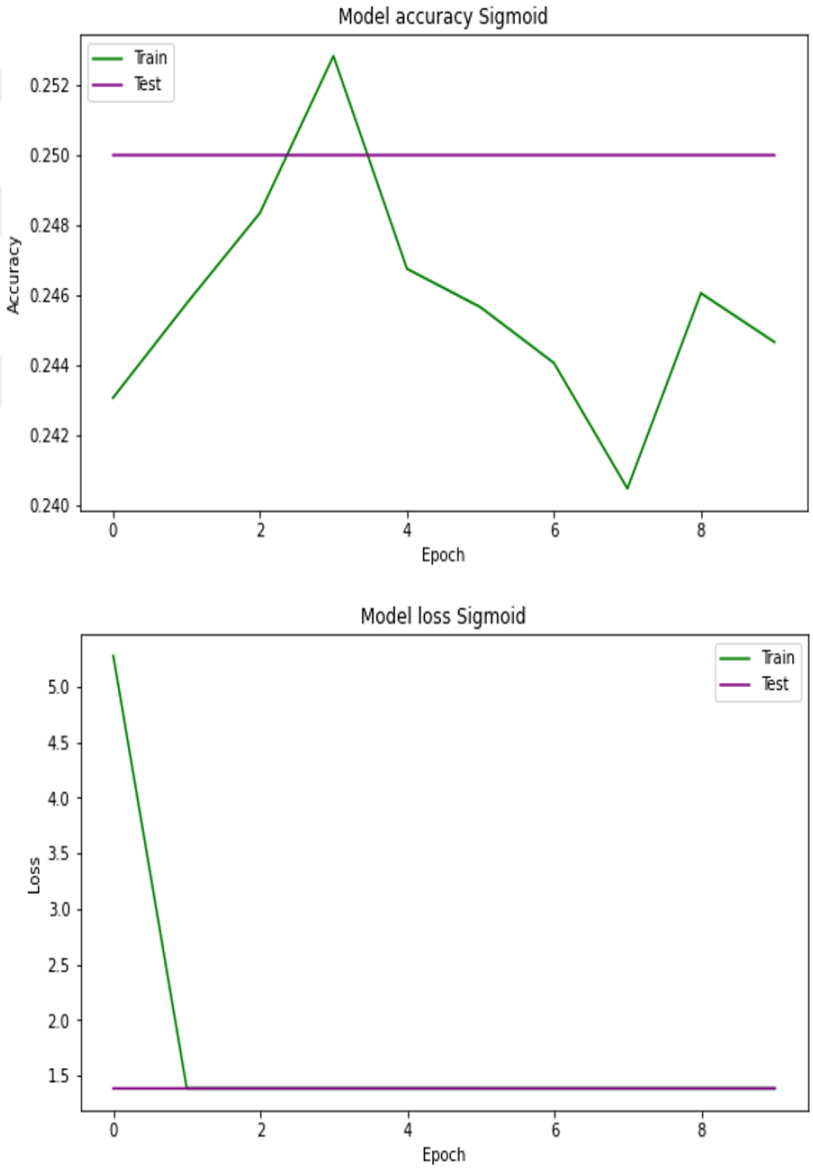
Öncelikle, filtre resim sol üst köşesine konumlandırılır. Daha sonra bu iki matrisin indisleri birbirleriyle ile çarpılır ve tüm sonuçlar toplanır. Hemen ardından filtre sağa 1 piksel kaydırılarak çarpma ve toplama işlemi tekrarlanır. Aynı işlemler tüm satırlar için tekrarlanarak hesaplanır. Bu işlemler sonucunda artık elimizde artık öznitelik haritası mevcuttur. Eğer sınıflandırılacak görüntü renkli ise RGB katmanları için bu işlemler tekrarlanır. Özetle filtreyi resim üzerinde hareket ettirerek ve matris çarpımını kullanarak özellikler tespit edilir. Bir görüntü üzerinde birden fazla convolutional katmanı kullanılır.

Tüm convolutional katmanlarından sonra genellikle doğrusal olmayan katman gelir. Buradaki en temel sorun doğrusallıktır. Burada kullanılan katman activation Layer (aktivasyon katmanı) olarak adlandırılır.

Önceleri sigmoid ve tahn gibi doğrusal olmayan fonksiyonlar sıklıkla kullanılırken günümüzde problemlerle de yakından ilgili sinir ağının eğitilmesi konusunda en iyi sonucu veren Rectifier (Relu) fonksiyonudur. Genellikle çıkış değil ara katmanlarda kullanılır. Çıkış için ise probleme göre sigmoid ya da Softmax tercih edilir (Nwankpa et al., 2018). Bu çalışmada belirlenecek aktivasyon fonksiyonu için en sık kullanılan aktivasyon fonksiyonları karşılaştırılıp sonuçlarına göre karar verilecektir.

2.6.2.2. Aktivasyon Fonksiyonlarının Karşılaştırılması

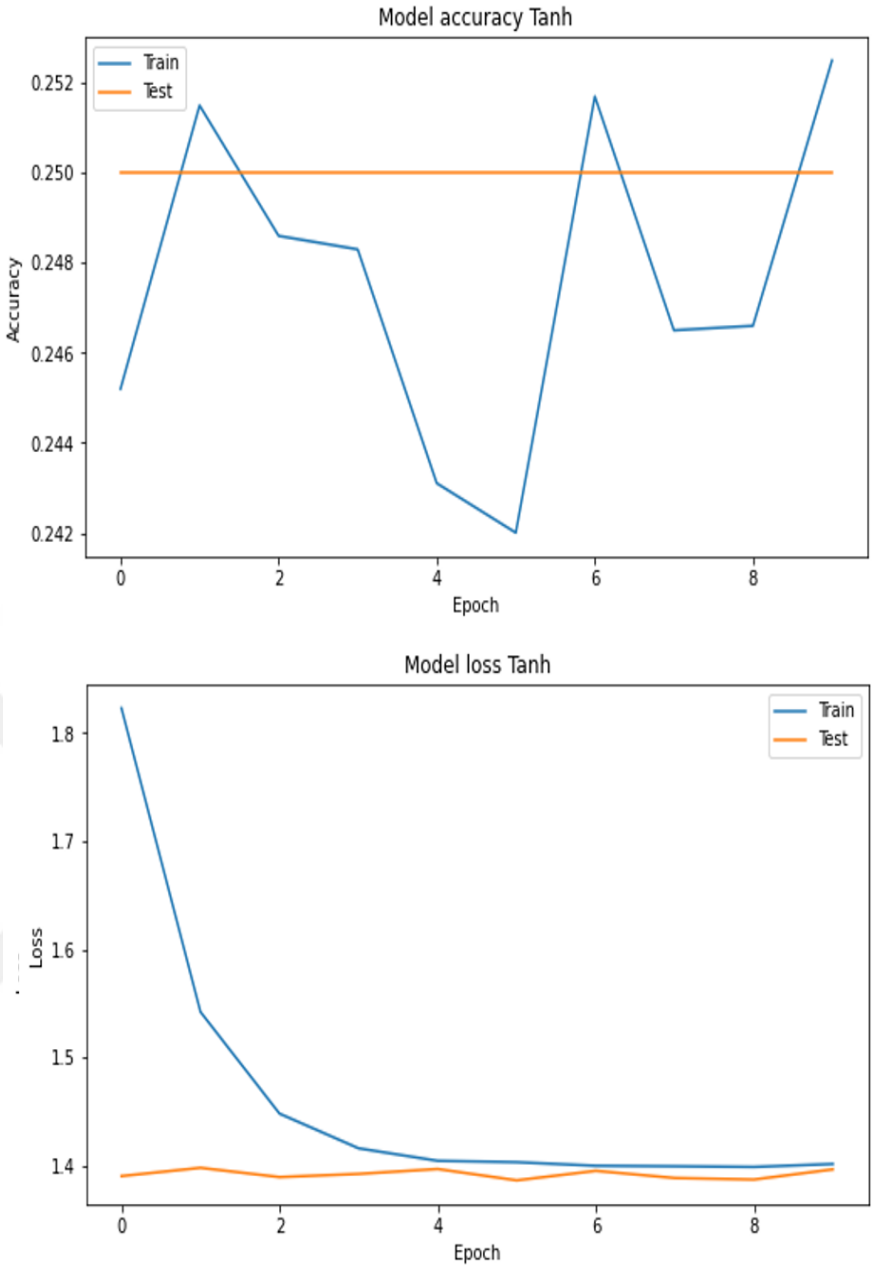
Bu çalışma için kurgulanan model içerisinde kullanılacak aktivasyon fonksiyonunu belirleyebilmek için en sık kullanılan 5 adet aktivasyon fonksiyonu irdelenmiş olup sonuçları grafik ve epoch satır değerleri ile verilmiştir. Karşılaştırılacak bu 5 adet fonksiyonun modelleri oluşturulurken eğitim ve test için kullanılan veri tabanındaki görüntülerin boyutu 600x450 piksel ölçülerinde orijinal boyutları girilmiş olup, 10 epoch ve batch_size 32 değeri verilerek yaklaşık 12 saat kısıtlı zaman desteği sunan Google Colab tarafından gerçekleştirilmiştir.



Şekil 2.6.2.2.1: Sigmoid fonksiyonu

Tablo 2.6.2.2.1: Sigmoid fonksiyonu eğitim ve test sonuçları

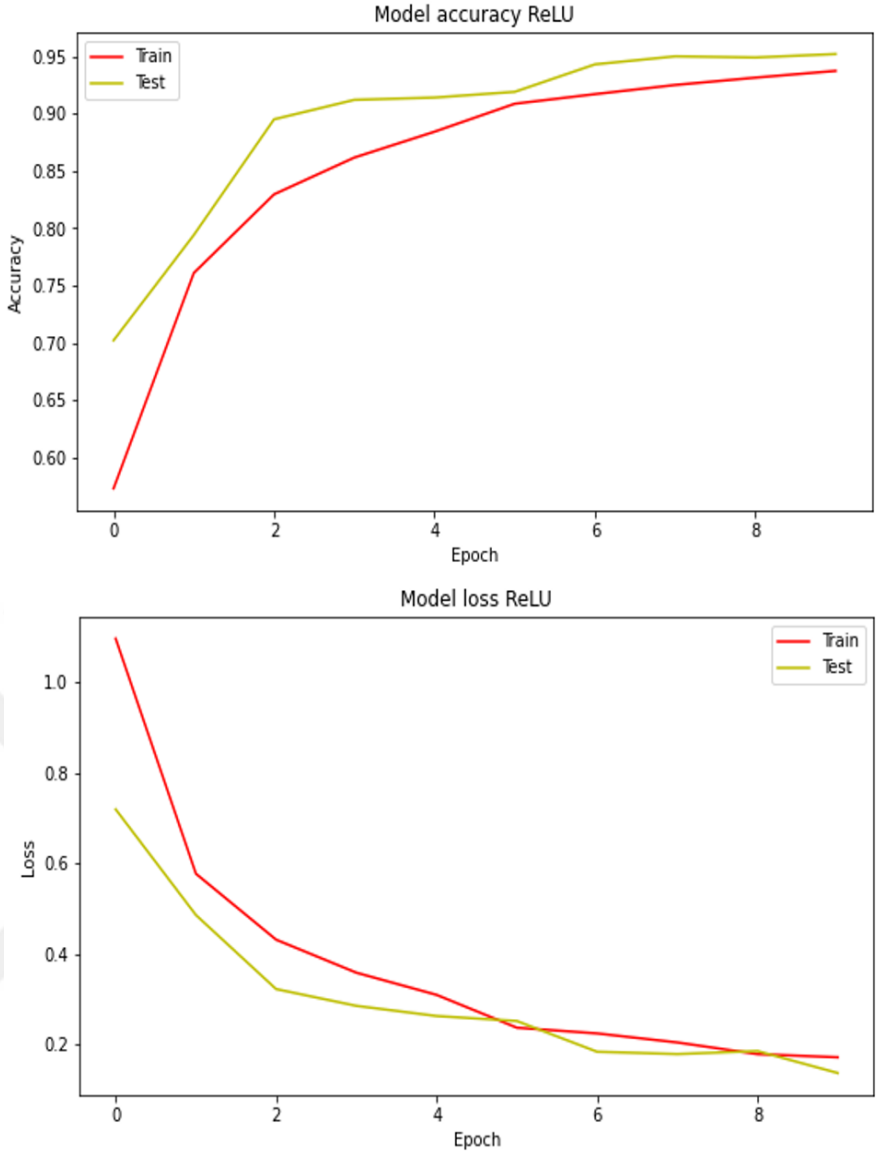
Epoch 1/10
313/312 [] - 8118s 26s/step - loss: 5.2760 - accuracy: 0.2431 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 2/10
313/312 [] - 657s 2s/step - loss: 1.3865 - accuracy: 0.2458 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 3/10
313/312 [] - 673s 2s/step - loss: 1.3865 - accuracy: 0.2484 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 4/10
313/312 [] - 695s 2s/step - loss: 1.3864 - accuracy: 0.2528 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 5/10
313/312 [] - 697s 2s/step - loss: 1.3865 - accuracy: 0.2468 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 6/10
313/312 [] - 668s 2s/step - loss: 1.3864 - accuracy: 0.2457 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 7/10
313/312 [] - 654s 2s/step - loss: 1.3864 - accuracy: 0.2441 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 8/10
313/312 [] - 662s 2s/step - loss: 1.3864 - accuracy: 0.2405 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 9/10
313/312 [] - 691s 2s/step - loss: 1.3864 - accuracy: 0.2461 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 10/10
313/312 [] - 680s 2s/step - loss: 1.3864 - accuracy: 0.2447 - val_loss: 1.3863 - val_accuracy: 0.2500
32/32 [] - 11s 359ms/step - loss: 1.3863 - accuracy: 0.2500
Test loss: 1.3863062858581543
Test accuracy: 0.25
313/313 [] - 612s 2s/step - loss: 1.3863 - accuracy: 0.2499
Training loss: 1.3863039016723633
Training accuracy: 0.2498500943183899



Şekil 2.6.2.2.2: Tanh fonksiyonu

Tablo 2.6.2.2.2: Tanh fonksiyonu eğitim ve test sonuçları

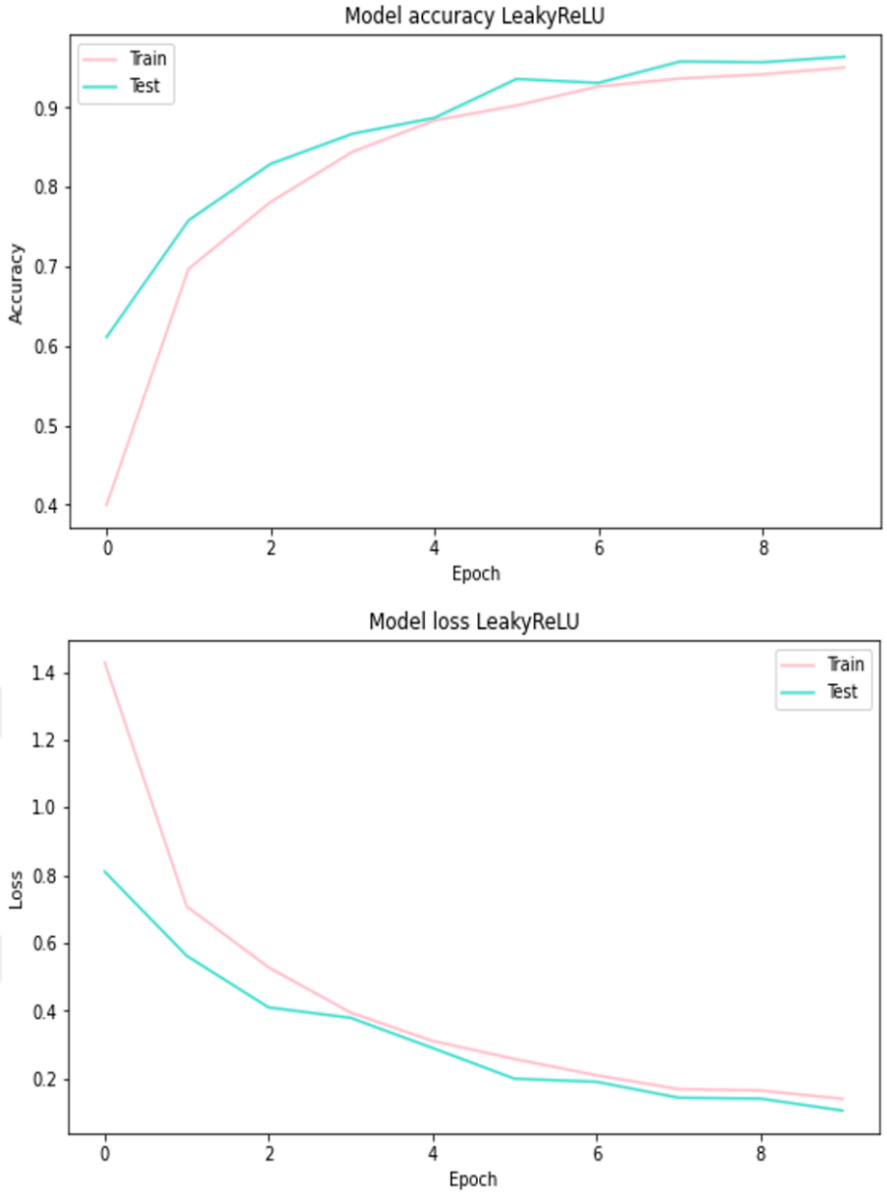
Epoch 1/10
313/312 [] - 4386s 14s/step - loss: 1.8231 - accuracy: 0.2452 - val_loss: 1.3905 - val_accuracy: 0.2500
Epoch 2/10
313/312 [] - 571s 2s/step - loss: 1.5422 - accuracy: 0.2515 - val_loss: 1.3981 - val_accuracy: 0.2500
Epoch 3/10
313/312 [] - 567s 2s/step - loss: 1.4482 - accuracy: 0.2486 - val_loss: 1.3895 - val_accuracy: 0.2500
Epoch 4/10
313/312 [] - 567s 2s/step - loss: 1.4161 - accuracy: 0.2483 - val_loss: 1.3925 - val_accuracy: 0.2500
Epoch 5/10
313/312 [] - 570s 2s/step - loss: 1.4046 - accuracy: 0.2431 - val_loss: 1.3970 - val_accuracy: 0.2500
Epoch 6/10
313/312 [] - 572s 2s/step - loss: 1.4033 - accuracy: 0.2420 - val_loss: 1.3865 - val_accuracy: 0.2500
Epoch 7/10
313/312 [] - 568s 2s/step - loss: 1.4000 - accuracy: 0.2517 - val_loss: 1.3953 - val_accuracy: 0.2500
Epoch 8/10
313/312 [] - 575s 2s/step - loss: 1.3996 - accuracy: 0.2465 - val_loss: 1.3887 - val_accuracy: 0.2500
Epoch 9/10
313/312 [] - 569s 2s/step - loss: 1.3989 - accuracy: 0.2466 - val_loss: 1.3873 - val_accuracy: 0.2500
Epoch 10/10
313/312 [] - 574s 2s/step - loss: 1.4016 - accuracy: 0.2525 - val_loss: 1.3967 - val_accuracy: 0.2500
32/32 [] - 10s 318ms/step - loss: 1.3967 - accuracy: 0.2500
Test loss: 1.3966723680496216
Test accuracy: 0.25
313/313 [] - 543s 2s/step - loss: 1.3966 - accuracy: 0.2498
Training loss: 1.3965814113616943
Training accuracy: 0.2498001605272293



Şekil 2.6.2.2.3: ReLU fonksiyonu

Tablo 2.6.2.2.3: ReLU fonksiyonu eğitim ve test sonuçları

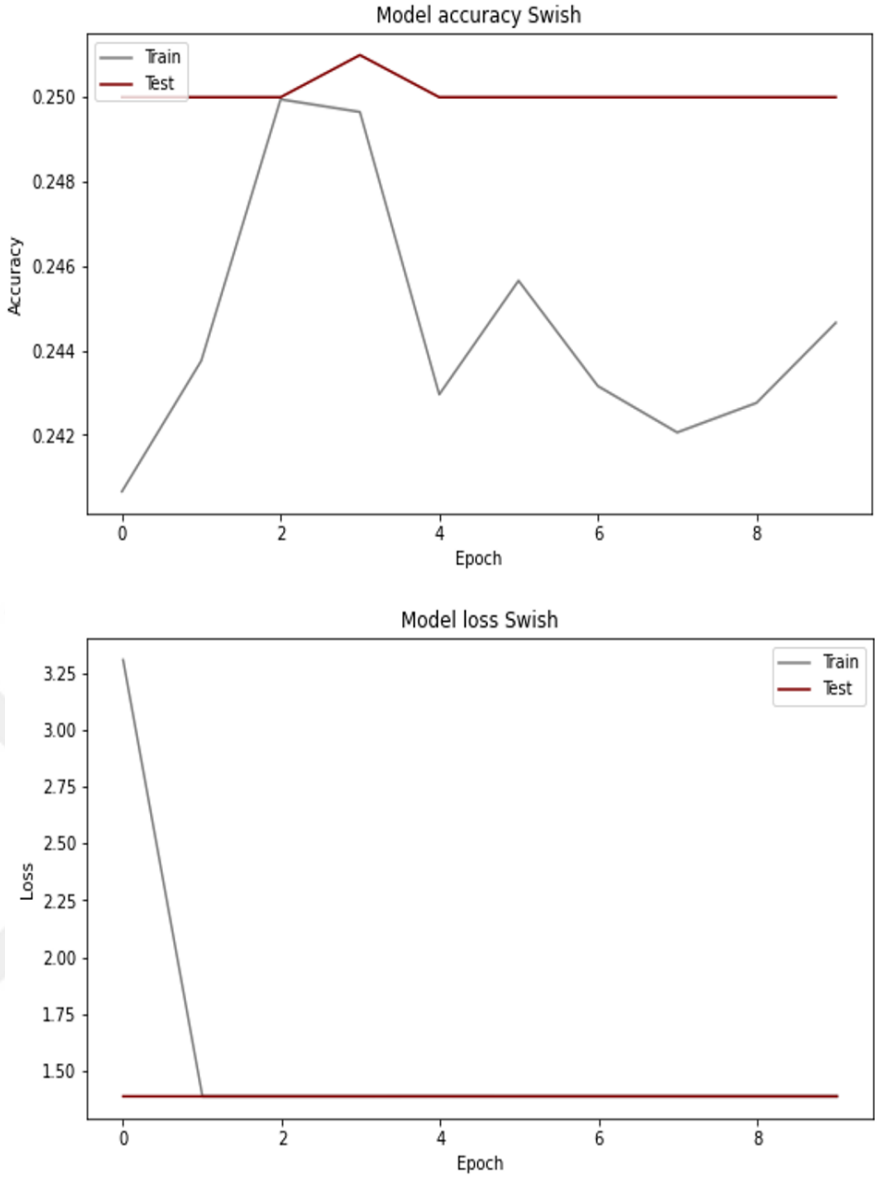
Epoch 1/10
313/312 [] - 5447s 17s/step - loss: 1.0968 - accuracy: 0.5727 - val_loss: 0.7192 - val_accuracy: 0.7020
Epoch 2/10
313/312 [] - 596s 2s/step - loss: 0.5768 - accuracy: 0.7608 - val_loss: 0.4855 - val_accuracy: 0.7940
Epoch 3/10
313/312 [] - 589s 2s/step - loss: 0.4309 - accuracy: 0.8295 - val_loss: 0.3217 - val_accuracy: 0.8950
Epoch 4/10
313/312 [] - 585s 2s/step - loss: 0.3580 - accuracy: 0.8616 - val_loss: 0.2846 - val_accuracy: 0.9120
Epoch 5/10
313/312 [] - 588s 2s/step - loss: 0.3089 - accuracy: 0.8842 - val_loss: 0.2621 - val_accuracy: 0.9140
Epoch 6/10
313/312 [] - 596s 2s/step - loss: 0.2363 - accuracy: 0.9086 - val_loss: 0.2511 - val_accuracy: 0.9190
Epoch 7/10
313/312 [] - 608s 2s/step - loss: 0.2237 - accuracy: 0.9170 - val_loss: 0.1831 - val_accuracy: 0.9430
Epoch 8/10
313/312 [] - 605s 2s/step - loss: 0.2037 - accuracy: 0.9250 - val_loss: 0.1780 - val_accuracy: 0.9500
Epoch 9/10
313/312 [] - 609s 2s/step - loss: 0.1779 - accuracy: 0.9314 - val_loss: 0.1846 - val_accuracy: 0.9490
Epoch 10/10
313/312 [] - 611s 2s/step - loss: 0.1709 - accuracy: 0.9373 - val_loss: 0.1361 - val_accuracy: 0.9520
32/32 [] - 10s 325ms/step - loss: 0.1361 - accuracy: 0.9520
Test loss: 0.1360524296760559
Test accuracy: 0.9520000219345093
313/313 [] - 583s 2s/step - loss: 0.0940 - accuracy: 0.9656
Training loss: 0.0939832478761673
Training accuracy: 0.9656000137329102



Şekil 2.6.2.2.4: LeakyReLU fonksiyonu

Tablo 2.6.2.2.4: LeakyReLU fonksiyonu eğitim ve test sonuçları

Epoch 1/10
313/312 [] - 606s 2s/step - loss: 1.4278 - accuracy: 0.3998 - val_loss: 0.8106 - val_accuracy: 0.6110
Epoch 2/10
313/312 [] - 607s 2s/step - loss: 0.7081 - accuracy: 0.6969 - val_loss: 0.5623 - val_accuracy: 0.7580
Epoch 3/10
313/312 [] - 604s 2s/step - loss: 0.5281 - accuracy: 0.7810 - val_loss: 0.4099 - val_accuracy: 0.8290
Epoch 4/10
313/312 [] - 617s 2s/step - loss: 0.3948 - accuracy: 0.8443 - val_loss: 0.3792 - val_accuracy: 0.8670
Epoch 5/10
313/312 [] - 610s 2s/step - loss: 0.3110 - accuracy: 0.8838 - val_loss: 0.2901 - val_accuracy: 0.8870
Epoch 6/10
313/312 [] - 608s 2s/step - loss: 0.2578 - accuracy: 0.9026 - val_loss: 0.1995 - val_accuracy: 0.9360
Epoch 7/10
313/312 [] - 602s 2s/step - loss: 0.2098 - accuracy: 0.9266 - val_loss: 0.1905 - val_accuracy: 0.9310
Epoch 8/10
313/312 [] - 600s 2s/step - loss: 0.1690 - accuracy: 0.9366 - val_loss: 0.1435 - val_accuracy: 0.9580
Epoch 9/10
313/312 [] - 600s 2s/step - loss: 0.1649 - accuracy: 0.9418 - val_loss: 0.1407 - val_accuracy: 0.9570
Epoch 10/10
313/312 [] - 600s 2s/step - loss: 0.1401 - accuracy: 0.9504 - val_loss: 0.1054 - val_accuracy: 0.9640
32/32 [] - 11s 340ms/step - loss: 0.1054 - accuracy: 0.9640
Test loss: 0.10542885959148407
Test accuracy: 0.9639999866485596
313/313 [] - 571s 2s/step - loss: 0.0674 - accuracy: 0.9767
Training loss: 0.06737193465232849
Training accuracy: 0.9767000079154968



Şekil 2.6.2.2.5: Swish fonksiyonu

Tablo 2.6.2.2.5: Swish fonksiyonu eğitim ve test sonuçları

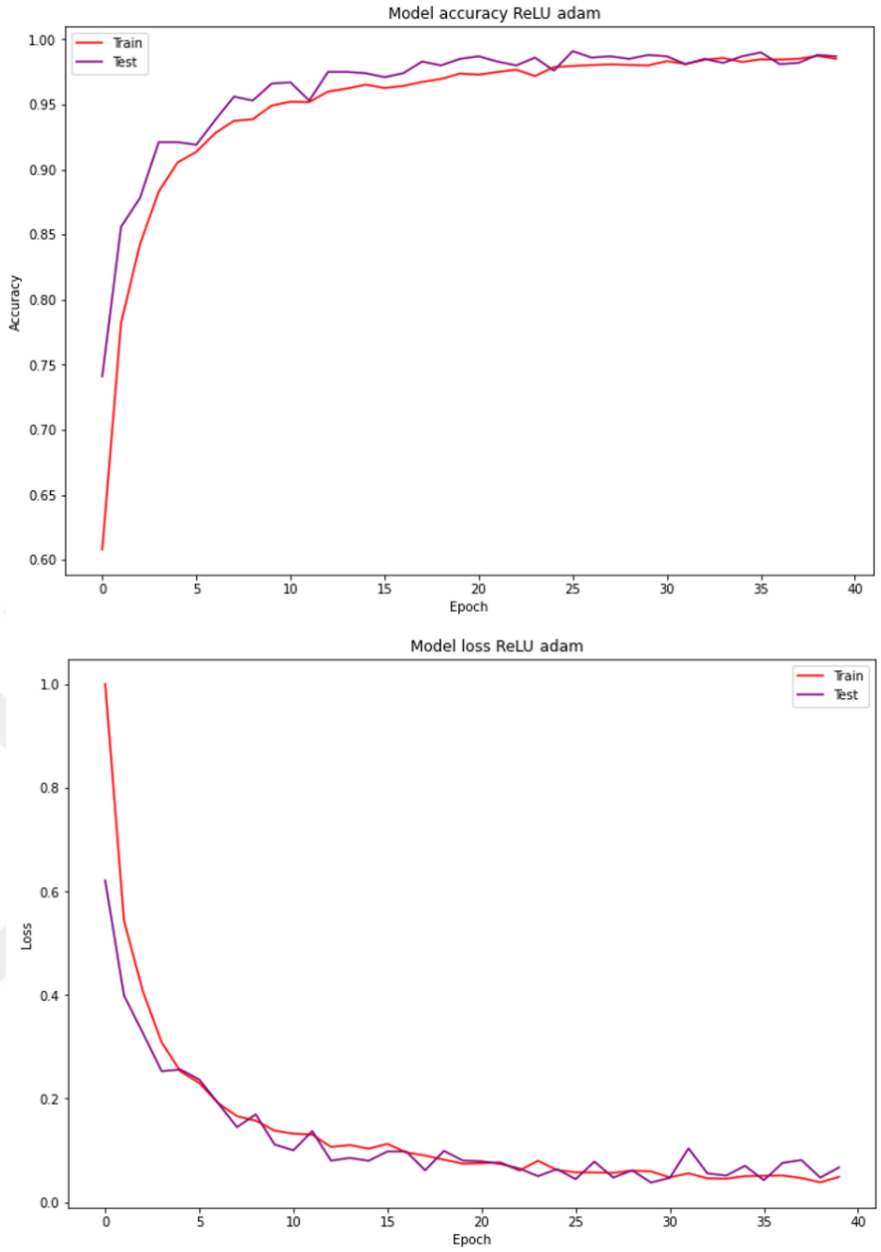
Epoch 1/10
313/312 [] - 8581s 27s/step - loss: 3.3085 - accuracy: 0.2407 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 2/10
313/312 [] - 711s 2s/step - loss: 1.3864 - accuracy: 0.2438 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 3/10
313/312 [] - 717s 2s/step - loss: 1.3864 - accuracy: 0.2500 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 4/10
313/312 [] - 701s 2s/step - loss: 1.3864 - accuracy: 0.2497 - val_loss: 1.3862 - val_accuracy: 0.2510
Epoch 5/10
313/312 [] - 685s 2s/step - loss: 1.3866 - accuracy: 0.2430 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 6/10
313/312 [] - 683s 2s/step - loss: 1.3864 - accuracy: 0.2457 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 7/10
313/312 [] - 686s 2s/step - loss: 1.3864 - accuracy: 0.2432 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 8/10
313/312 [] - 691s 2s/step - loss: 1.3864 - accuracy: 0.2421 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 9/10
313/312 [] - 690s 2s/step - loss: 1.3864 - accuracy: 0.2428 - val_loss: 1.3863 - val_accuracy: 0.2500
Epoch 10/10
313/312 [] - 690s 2s/step - loss: 1.3864 - accuracy: 0.2447 - val_loss: 1.3863 - val_accuracy: 0.2500
32/32 [] - 12s 377ms/step - loss: 1.3863 - accuracy: 0.2500
Test loss: 1.3862993717193604
Test accuracy: 0.25
313/313 [] - 616s 2s/step - loss: 1.3863 - accuracy: 0.2499
Training loss: 1.3862980604171753
Training accuracy: 0.2498500943183899

Tablo 2.6.2.2.6: Beş fonksiyonun eğitim ve test sonuçlarının karşılaştırılması

Aktivasyon Fonksiyonları	Training Loss	Training Accuracy	Test Loss	Test Accuracy	Epoch
sigmoid	1,38630390	0,24985009	1,38630629	0,25000000	10
tanh	1,39658141	0,24980016	1,39667237	0,25000000	10
ReLU	0,09398325	0,96560001	0,13605243	0,95200002	10
LeakyReLU	0,06737193	0,97670001	0,10542886	0,96399999	10
swish	1,38629806	0,24985009	1,38629937	0,25000000	10

Tablo 2.6.2.2.6’ ya bakıldığında en iyi sonuçları ReLU ve LeakyReLU aktivasyon fonksiyonlarının verdiği görülmüştür. Ayrıca fonksiyonların train ve test değerleri birbirine oldukça yakın çıkmıştır. Bunun sebebi LeakyReLU fonksiyonunun ReLU fonksiyondan türetilmiş olmasıdır. Sonuçlardan yola çıkılarak iki fonksiyon tekrar 40 epoch değeri döndürülerek karşılaştırılmıştır. Karşılaştırma sonucu grafik ve epoch satır değerleri şekil 40 da görüldüğü gibidir. ReLU ve LeakyReLU fonksiyonları için train ve test değerlerine bakıldığında aralarında çok küçük miktarda farklılık olduğu görülmektedir. Gerçekleştirilen model eğitimlerinde anlık olarak alınan Google Colab tarafından sağlanan ücretsiz GPU donanım desteği, modelin eğitimine doğrudan etki yapabilmektedir.

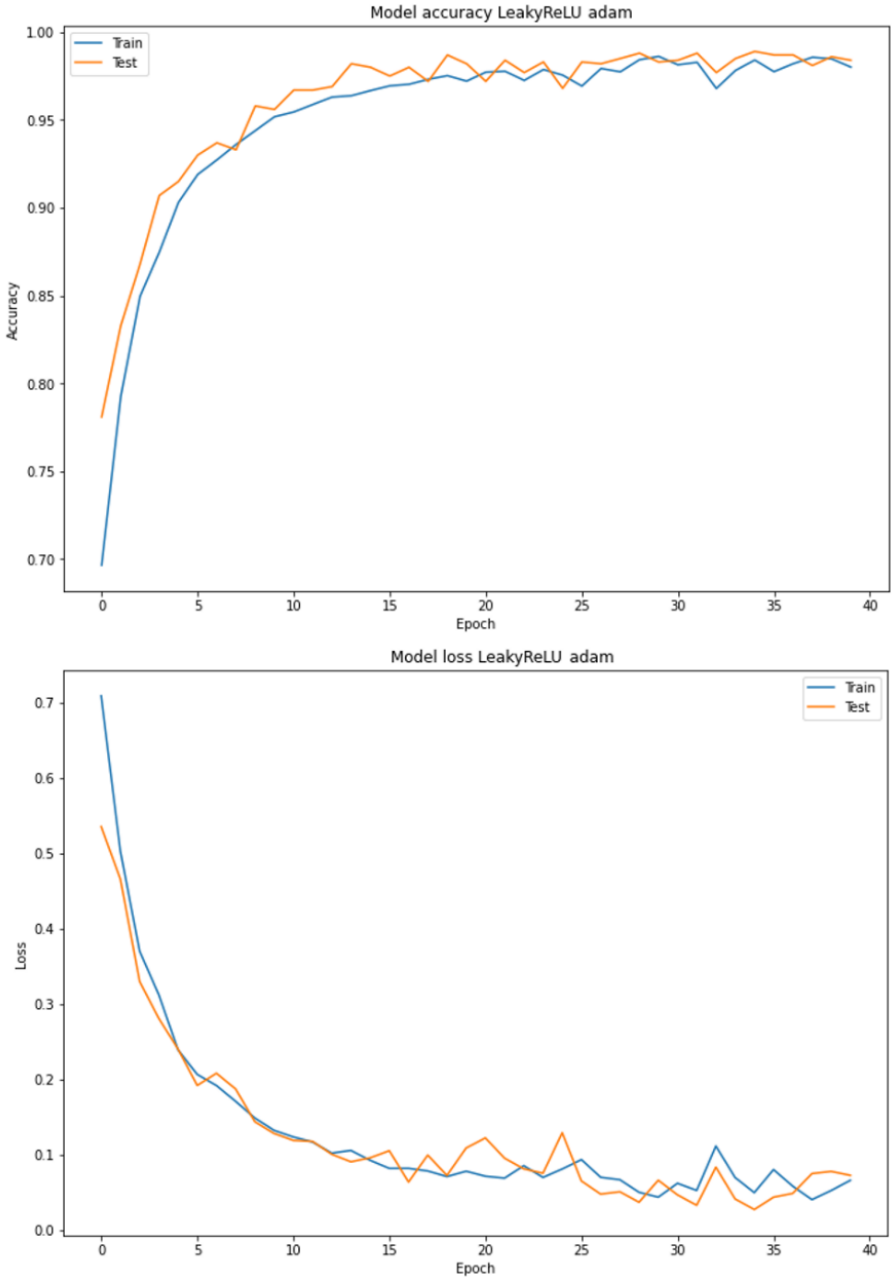
Bu iki fonksiyon sonuçları tekrar 40 epoch olarak döndürüldüğünde sonuçlarda küçükte olsa farklılıklar ortaya çıktığı görülmüştür. Örneğin GPU desteği anlık bir modeli eğitirken en üst seviye 1.70 GB kullanım imkânı verirken diğer eğitim esnasında 2.35 GB kullanım miktarı gibi destek sağladığı gözlemlenmiştir. Böylelikle eğitim sürelerinde de farklılıklar oluşmuştur. Bu sebepten dolayı birbirine yakın sonuçlar arasındaki farkın göz ardı edilmesi gerektiği izlenimi vermiştir.



Şekil 2.6.2.2.6: ReLU fonksiyonunu adam ile optimize edilmesi

Tablo 2.6.2.2.7: ReLU - adam ile optimize parametre değerleri

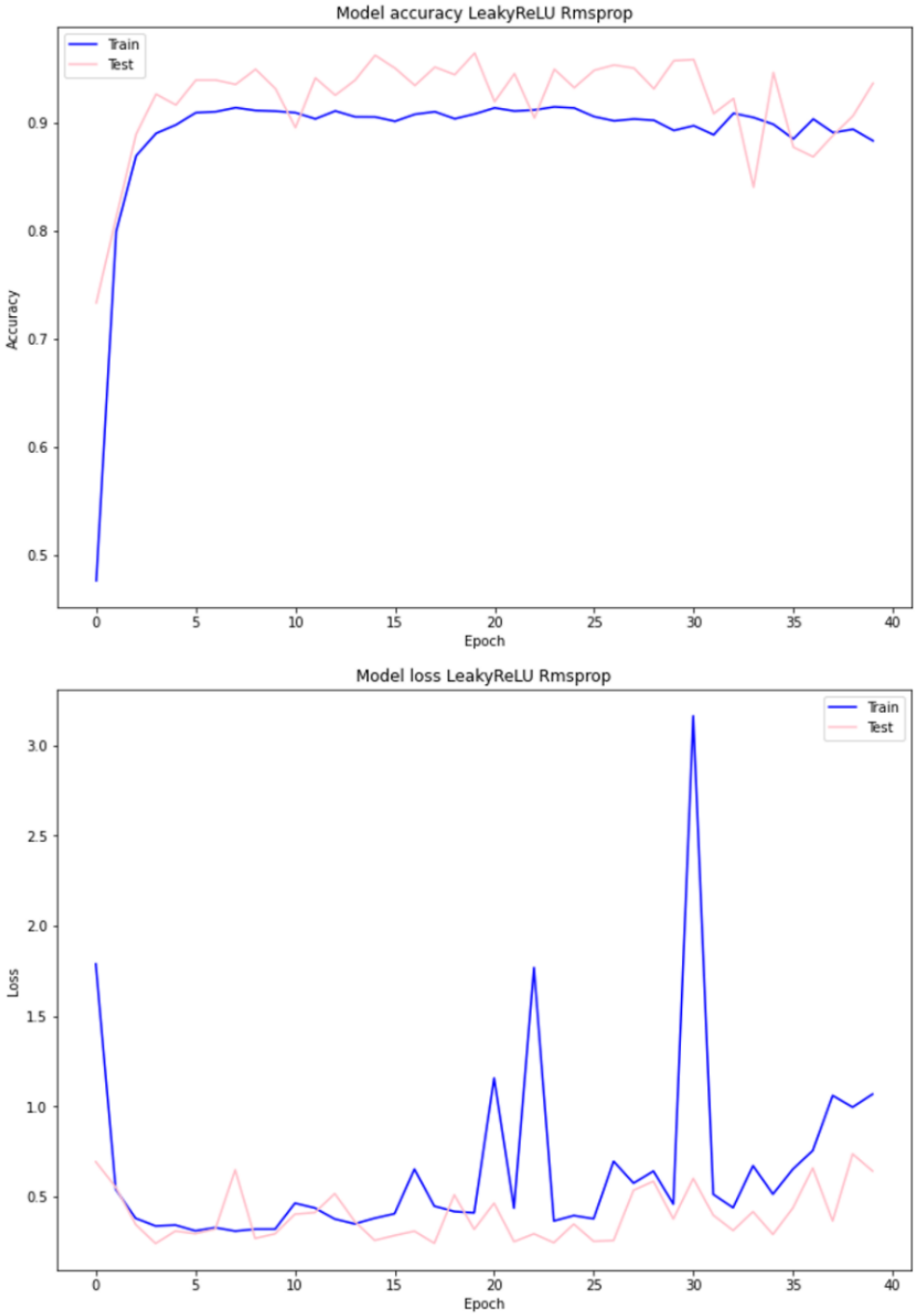
Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 598, 448, 32)	896
max_pooling2d (MaxPooling2D)	(None, 299, 224, 32)	0
conv2d_1 (Conv2D)	(None, 297, 222, 64)	18496
max_pooling2d_1 (MaxPooling2)	(None, 148, 111, 64)	0
conv2d_2 (Conv2D)	(None, 146, 109, 128)	73856
max_pooling2d_2 (MaxPooling2)	(None, 73, 54, 128)	0
flatten (Flatten)	(None, 504576)	0
dense (Dense)	(None, 128)	64585856
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Total params: 64,679,620		
Trainable params: 64,679,620		
Non-trainable params: 0		



Şekil 2.6.2.2.7: LeakyReLU fonksiyonunu adam ile optimize edilmesi

Tablo 2.6.2.2.8: LeakyReLU - adam ile optimize parametre değerleri

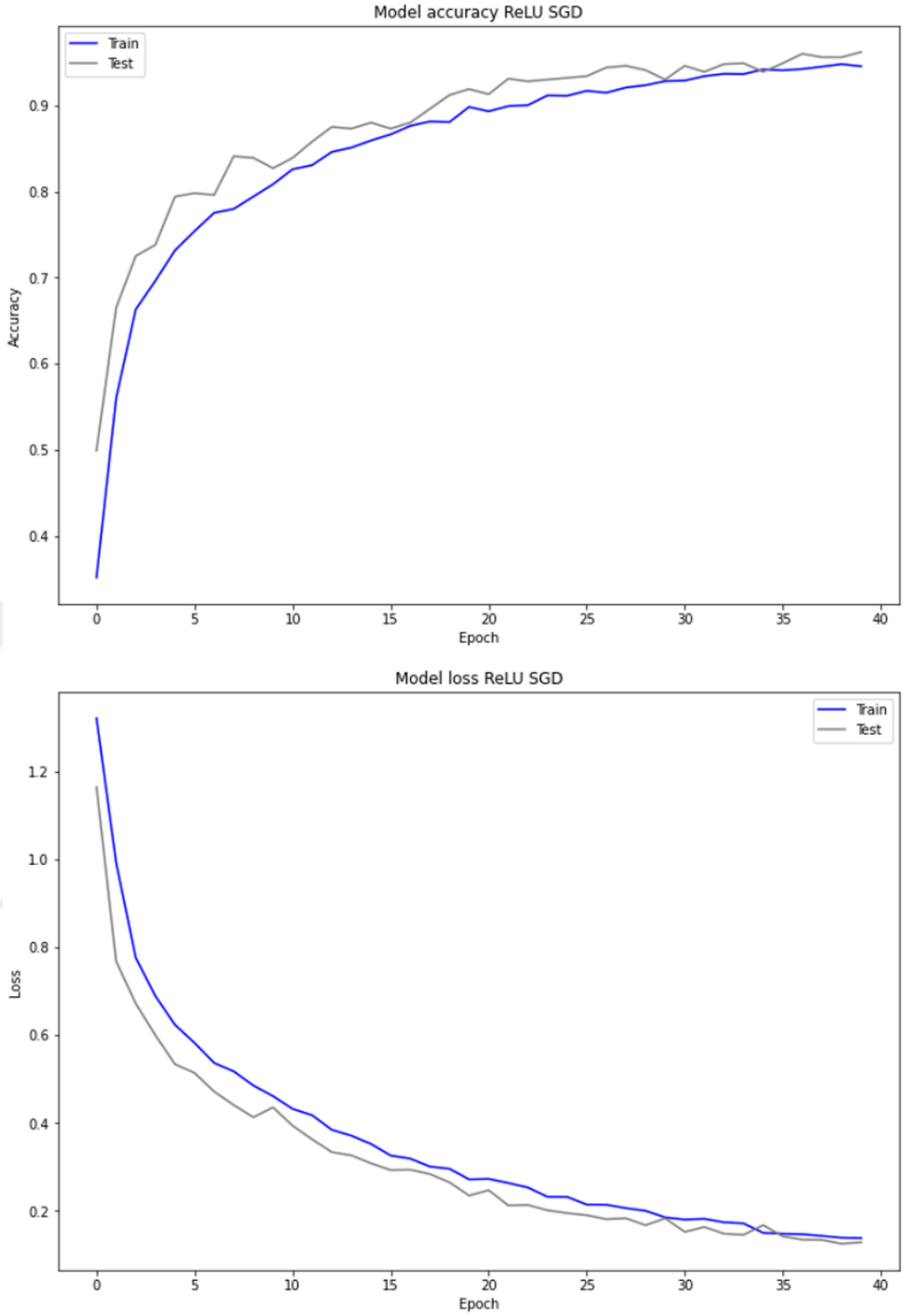
Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 598, 448, 32)	896
leaky_re_lu (LeakyReLU)	(None, 598, 448, 32)	0
max_pooling2d (MaxPooling2D)	(None, 299, 224, 32)	0
conv2d_1 (Conv2D)	(None, 297, 222, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 297, 222, 64)	0
max_pooling2d_1 (MaxPooling2)	(None, 148, 111, 64)	0
conv2d_2 (Conv2D)	(None, 146, 109, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 146, 109, 128)	0
max_pooling2d_2 (MaxPooling2)	(None, 73, 54, 128)	0
flatten (Flatten)	(None, 504576)	0
dense (Dense)	(None, 128)	64585856
leaky_re_lu_3 (LeakyReLU)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Total params: 64,679,620		
Trainable params: 64,679,620		
Non-trainable params: 0		



Şekil 2.6.2.2.8: LeakyReLU fonksiyonunu Rmsprop ile optimize edilmesi

Tablo 2.6.2.2.9: LeakyReLU - Rmsprop ile optimize parametre değerleri

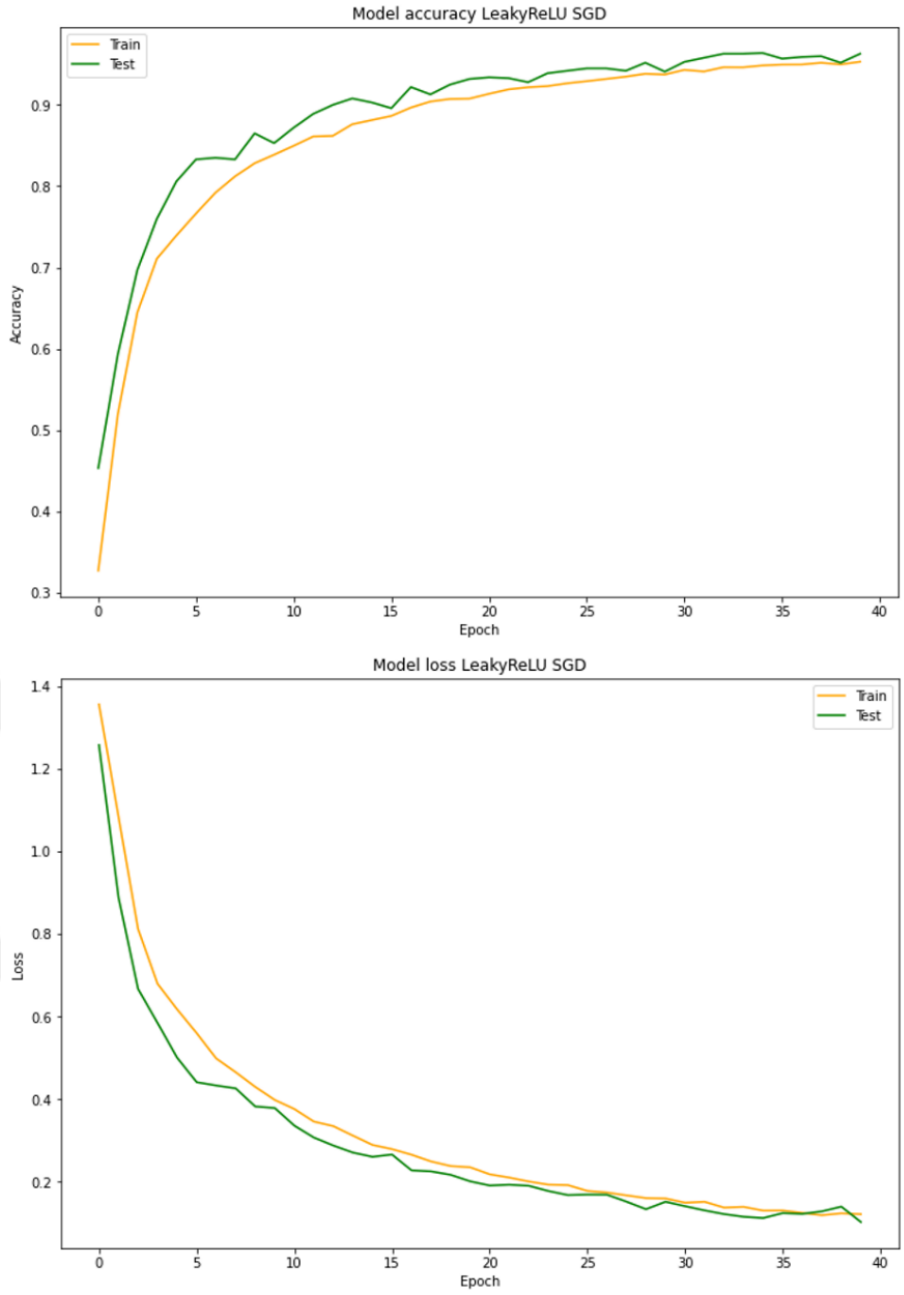
Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 598, 448, 32)	896
leaky_re_lu (LeakyReLU)	(None, 598, 448, 32)	0
max_pooling2d (MaxPooling2D)	(None, 299, 224, 32)	0
conv2d_1 (Conv2D)	(None, 297, 222, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 297, 222, 64)	0
max_pooling2d_1 (MaxPooling2)	(None, 148, 111, 64)	0
conv2d_2 (Conv2D)	(None, 146, 109, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 146, 109, 128)	0
max_pooling2d_2 (MaxPooling2)	(None, 73, 54, 128)	0
flatten (Flatten)	(None, 504576)	0
dense (Dense)	(None, 128)	64585856
leaky_re_lu_3 (LeakyReLU)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Total params: 64,679,620		
Trainable params: 64,679,620		
Non-trainable params: 0		



Şekil 2.6.2.2.9: ReLU fonksiyonunu SGD ile optimize edilmesi

Tablo 2.6.2.2.10: ReLU - SGD ile optimize parametre değerleri

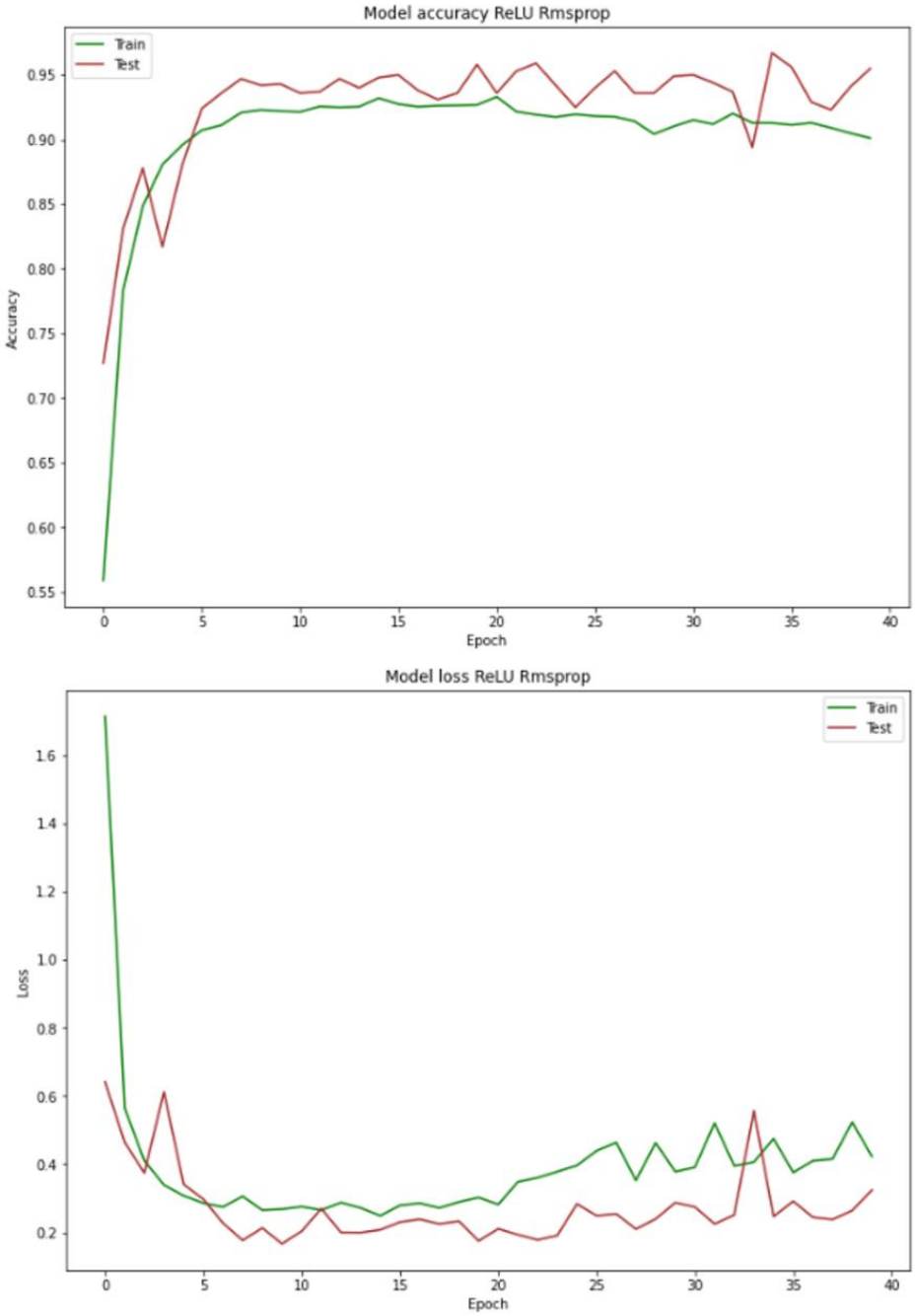
Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 598, 448, 32)	896
max_pooling2d (MaxPooling2D)	(None, 299, 224, 32)	0
conv2d_1 (Conv2D)	(None, 297, 222, 64)	18496
max_pooling2d_1 (MaxPooling2)	(None, 148, 111, 64)	0
conv2d_2 (Conv2D)	(None, 146, 109, 128)	73856
max_pooling2d_2 (MaxPooling2)	(None, 73, 54, 128)	0
flatten (Flatten)	(None, 504576)	0
dense (Dense)	(None, 128)	64585856
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Total params: 64,679,620		
Trainable params: 64,679,620		
Non-trainable params: 0		



Şekil 2.6.2.2.10: LeakyReLU fonksiyonunu SGD ile optimize edilmesi

Tablo 2.6.2.2.11: LeakyReLU - SGD ile optimize parametre değerleri

Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 598, 448, 32)	896
leaky_re_lu (LeakyReLU)	(None, 598, 448, 32)	0
max_pooling2d (MaxPooling2D)	(None, 299, 224, 32)	0
conv2d_1 (Conv2D)	(None, 297, 222, 64)	18496
leaky_re_lu_1 (LeakyReLU)	(None, 297, 222, 64)	0
max_pooling2d_1 (MaxPooling2)	(None, 148, 111, 64)	0
conv2d_2 (Conv2D)	(None, 146, 109, 128)	73856
leaky_re_lu_2 (LeakyReLU)	(None, 146, 109, 128)	0
max_pooling2d_2 (MaxPooling2)	(None, 73, 54, 128)	0
flatten (Flatten)	(None, 504576)	0
dense (Dense)	(None, 128)	64585856
leaky_re_lu_3 (LeakyReLU)	(None, 128)	0
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Total params: 64,679,620		
Trainable params: 64,679,620		
Non-trainable params: 0		



Şekil 2.6.2.2.11: ReLU fonksiyonunu Rmsprop ile optimize edilmesi

Tablo 2.6.2.2.12: ReLU - Rmsprop ile optimize parametre değerleri

Model: "sequential"		
Layer (type)	**Output Shape**	**Param #**
conv2d (Conv2D)	(None, 598, 448, 32)	896
max_pooling2d (MaxPooling2D)	(None, 299, 224, 32)	0
conv2d_1 (Conv2D)	(None, 297, 222, 64)	18496
max_pooling2d_1 (MaxPooling2)	(None, 148, 111, 64)	0
conv2d_2 (Conv2D)	(None, 146, 109, 128)	73856
max_pooling2d_2 (MaxPooling2)	(None, 73, 54, 128)	0
flatten (Flatten)	(None, 504576)	0
dense (Dense)	(None, 128)	64585856
dropout (Dropout)	(None, 128)	0
dense_1 (Dense)	(None, 4)	516
Total params: 64,679,620		
Trainable params: 64,679,620		
Non-trainable params: 0		

Tablo 2.6.2.2.13: ReLU - LeakyReLU optimize değerlerinin karşılaştırılması

Batch Size = 32		**ReLU**			**LeakyReLU**		
		adam	***rmsprop***	***sgd***	***adam***	***rmsprop***	***sgd***
1.Epoch	loss	1,0009	1,7145	1,3213	0.7085	1,7896	1,3565
	accuracy	0.6079	0.5590	0.3514	0.6966	0.4760	0.3269
	val_loss	0.6212	0.6421	1,1647	0.5350	0.6942	12.578
	val_ accuracy	0.7410	0.7270	0.4990	0.7810	0.7330	0.4530
10.Epoch	loss	0.1379	0.2682	0.4615	0.1319	0.3209	0.3985
	accuracy	0.9491	0.9220	0.8084	0.9518	0.9103	0.8388
	val_loss	0.1112	0.1668	0.4359	0.1277	0.2951	0.3789
	val_ accuracy	0.9660	0.9430	0.8270	0.9560	0.9310	0.8530
20.Epoch	loss	0.0743	0.3024	0.2721	0.0776	0.4119	0.2354
	accuracy	0.9737	0.9268	0.8982	0.9721	0.9077	0.9078
	val_loss	0.0803	0.1751	0.2353	0.1085	0.3203	0.2015

	val_ accuracy	0.9850	0.9580	0.9190	0.9820	0.9640	0.9320
30.Epoch	loss	0.0591	0.3784	0.1856	0.0432	0.4597	0.1598
	accuracy	0.9800	0.9103	0.9282	0.9862	0.8924	0.9374
	val_loss	0.0378	0.2873	0.1835	0.0657	0.3775	0.1520
	val_ accuracy	0.9880	0.9490	0.9300	0.9830	0.9570	0.9410
40.Epoch	loss	0.0487	0.4230	0.1384	0.0656	1,0684	0.1219
	accuracy	0.9851	0.9012	0.9454	0.9801	0.8829	0.9532
	val_loss	0.0670	0.3241	0.1290	0.0722	0.6430	0.1030
	val_ accuracy	0.9870	0.9550	0.9620	0.9840	0.9360	0.9630
Eğitim/Test Sonuçları	Training Loss	0.0134	0.1525	0.0672	0.0195	0.2069	0.0517
	Training Acc.	0.9959	0.9582	0.9750	0.9947	0.9369	0.9832
	Test Loss	0.0670	0.3241	0.1290	0.0722	0.6430	0.1030
	Test Acc.	0.9870	0.9550	0.9620	0.9840	0.9360	0.9630

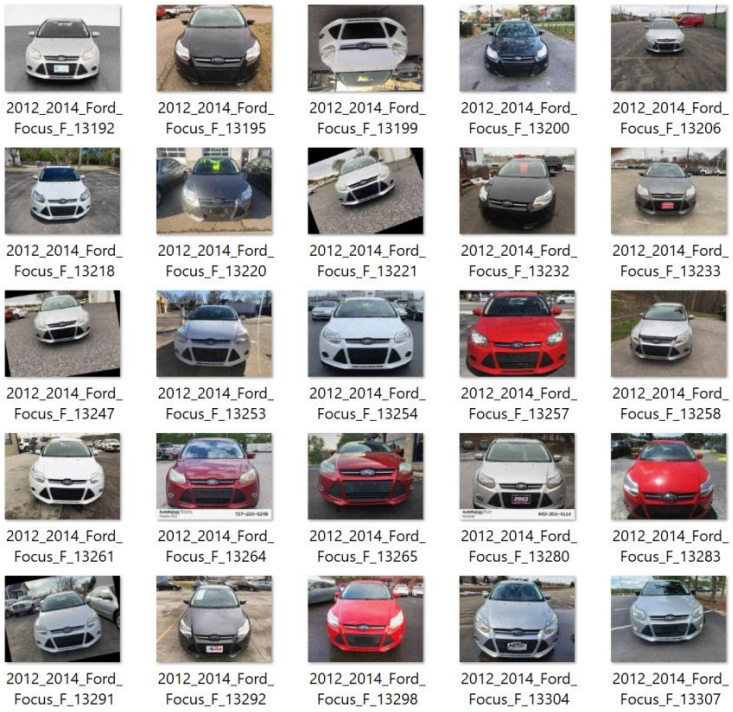
Tablo 2.6.2.2.13'e bakıldığında en başarılı eğitim ve test sonuçları için ara(gizli) katmanlarda ReLU aktivasyon fonksiyonunu verirken çıkış katmanında optimizasyon yöntemi için adam tercih edilmiştir. Çıkış katmanında ise softmax aktivasyon fonksiyonu kullanılmıştır.

Ayrıca bu parametrelerin seçilmesi problemle alakalı olarak değişiklik gösterse de literatür araştırmasında nöron sayısı 32 ile test edilen ReLU aktivasyon fonksiyonu ve optimizasyon yöntemi olarak "adam" en yüksek 5 başarı oranı veren listede olduğu gözlemlenmiştir (Ser & Bati, 2019). Modelde kullanılan "adam" optimizasyon yöntemi az bellek gereksinimi olan birinci dereceden gradyanlar gerektiren stokastik optimizasyon yöntemidir (Kingma & Ba, 2015).

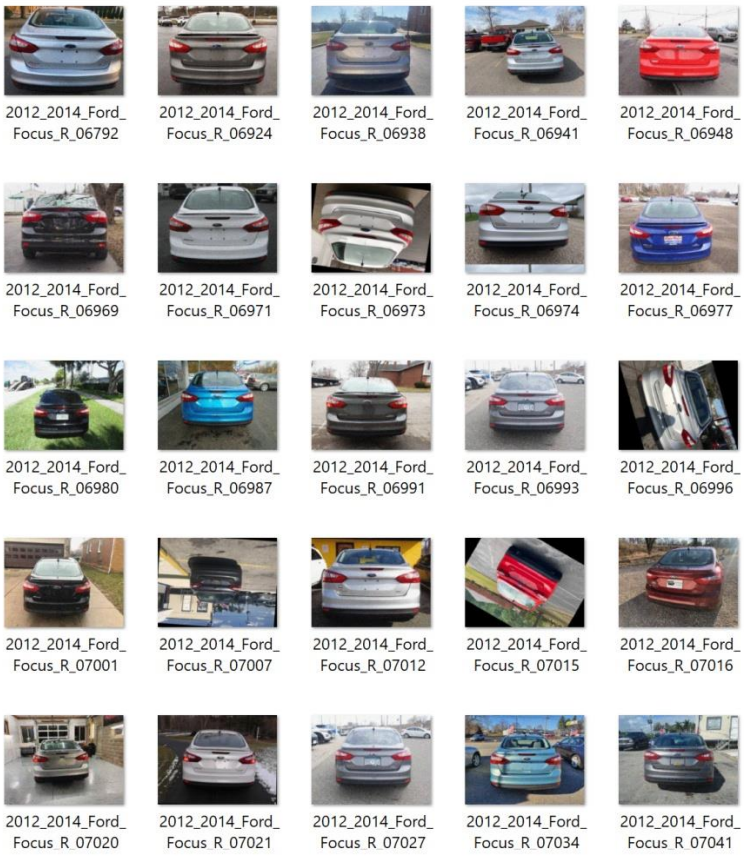
Belirlenen parametreler ile eğitilen modeli, eğitim ve test aşamasındaki görüntülerden ayrı olarak oluşturulan veri seti içinden rastgele seçilen resimler üzerinde gerçekleştiğimizde tablo de de görüldüğü üzere her sınıf için %100 doğruluk oranı gibi çok iyi sonuç verdiği görülmüştür.

Tablo 2.6.2.2.14: Rastgele 100 adet görüntü için test sonuçları

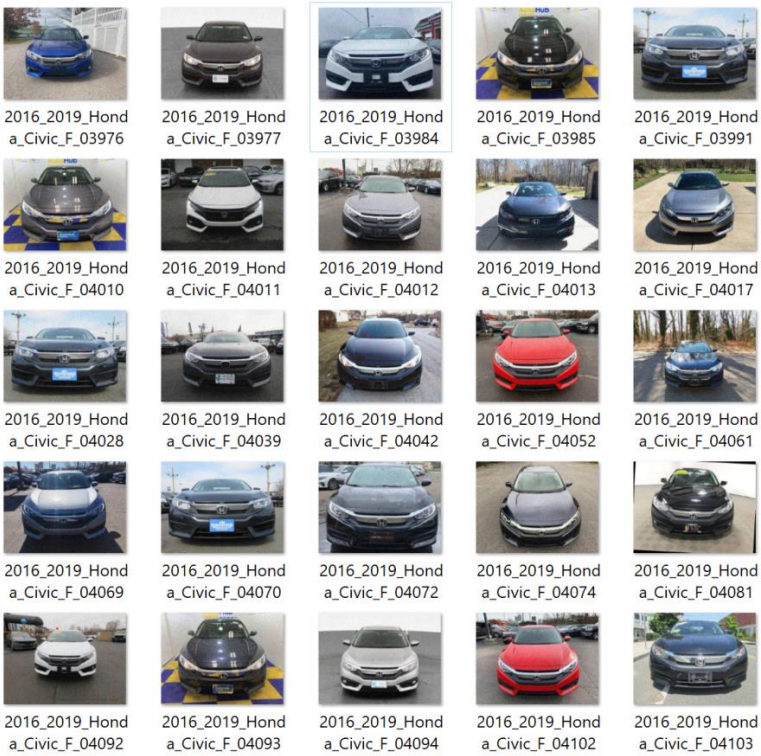
Marka/Model	**Rastgele Seçilen Görüntüler**	**Doğru Tespit**	**Yanlış Tespit**	Doğruluk Oranı
2012_2014_Ford_Focus_F	25	25	0	100%
2012_2014_Ford_Focus_R	25	25	0	100%
2016_2019_Honda_Civic_F	25	25	0	100%
2016_2019_Honda_Civic_R	25	25	0	100%



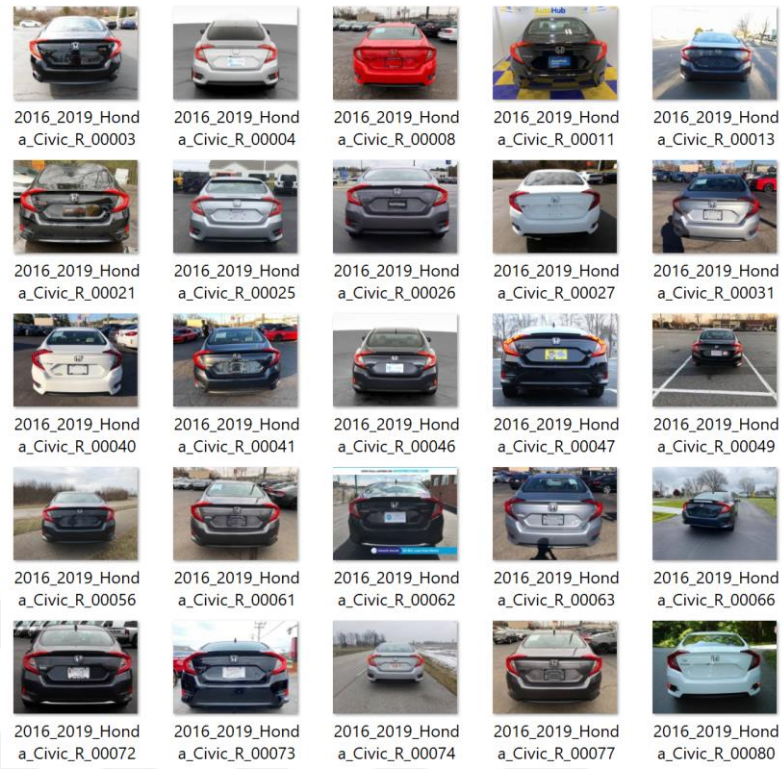
Şekil 2.6.2.2.12: 2012_2014_Ford_Focus_F test için rastgele 25 adet görüntü



Şekil 2.6.2.2.13: 2012_2014_Ford_Focus_R test için rastgele 25 adet görüntü



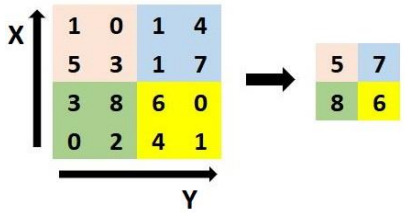
Şekil 2.6.2.2.14: 2016_2019_Honda_Civic_F test için rastgele 25 adet görüntü



Şekil 2.6.2.2.15: 2016_2019_Honda_Civic_R test için rastgele 25 adet görüntü

2.6.2.3. Havuzlama (Pooling)

Bu işlem genellikle ardışık evrişim (Convolutional) katmanlarından hemen sonra eklenilerek kullanılır. Ağın bu katmanında öğrenilen parametre yoktur. Amaç giriş matrisinin kanal sayısını sabit tutarak yükseklik ve genişlik bilgisini azaltarak hesaplama karmaşıklığını en aza indirgemek için kullanılan bir adımdır. Bu işlem için Şekil 2.34' de gösterilen örnek gibi, maksimum havuzlama işlemi uygulanır.

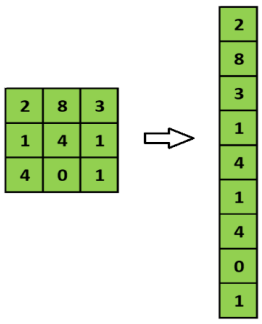


Şekil 2.6.2.3.1: Havuzlama işlemi

2.6.2.4. Düzleştirme Katmanı (Flattening Layer)

Bu katmanda ise son katman olan düzleştirme katmanı (Full Connected Layer)' in girişindeki verileri hazırlama işlemi gerçekleşir. Bu katman ise Convolutional ve

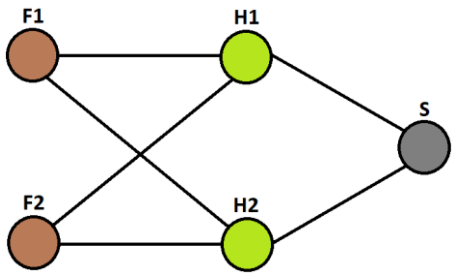
Pooling katmanlarından gelen matrislerin tek boyutlu diziye dönüştürülmüş verileri içerir.



Şekil 2.6.2.4.1: Tek boyutlu matrise çevirme

2.6.2.5. Tamamen Bağlı Katman (Fully Connected Layer)

Bu katmanın inputları Flattening' in oluşturduğu matrislerdir. Sonuç katmanı olarak bilinir. Yapay sinir ağının Flattening çıkışından gelen değeri giriş katmanı olarak alır. Daha sonra bu input değerleri ile gizli katmanlardan aldığı değerleri aktivasyon forksiyonuna göre çıkış değeri üreten katmandır. Şekil 2.6.2.5.1' de gösterilen basitleştirilmiş bir full connected layer görüntüsüdür. Flattening oluşturduğu F1 ve F2 değerleri ile Full Connected Layer için input değeri oluşturulur. Bu input değerleri ile Hidden Layer yani gizli katmanlar H1 ve H2 ile model tarafından belirlenen katsayılarla işleme girer. Bu işlem sonucunda belirlediğimiz softmax aktivasyon fonksiyonu ile S değeri yani sonuç tahmini üretilir.



Şekil 2.6.2.5.1: Tamamen bağlı katman örneği

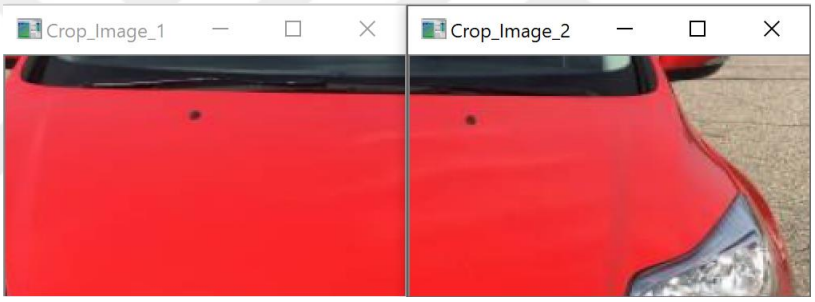
Algoritmanın akışında kullanılan parametrelerden bir diğeri ise Full Connected Layet katmanında kullanılan Dropout' dır. Bu parametre eğitim sırasında aşırı öğrenme(overfitting) olayını engellemek için bazı nöronları unutulmasını sağlar (Srivastava et al., 2018). Veri seti sayısının az olduğu durumlarda veya çok uzun

süreli bir eğitim yapılıyor ise aşırı öğrenmenin söz konusu olduğu unutulmamalıdır. Modelimiz ardışık olarak 32, 64, 128 birimlik (128 çıkışlı) ve ReLu ile aktive olan katmanlardan oluşmaktadır. Bu sebepten dolayı Full Connected Layer oluşturulurken 128 çıkışlı ReLU fonksiyonu kullanılır. Ama bizim çıkışımızda elimizde 4 adet sınıf bulunmaktadır. Bu nedenle çıkışta 4 değer olmasını istiyoruz. Bu sorunu çözmek için yeni bir katman eklemek yeterli olacaktır. Fakat yeni katmanı eklemeden önce dropout değeri 0,5 olarak belirlenerek ağırlıkların yarıya düşürülmesi ve aşırı öğrenmenin önüne geçilmesi amaçlanır. Daha sonra dense değeri 4 tane çıkışı olan yeni bir katman ekleyip aktivasyon fonksiyonunu softmax yapıyoruz.

ÜÇÜNCÜ BÖLÜM

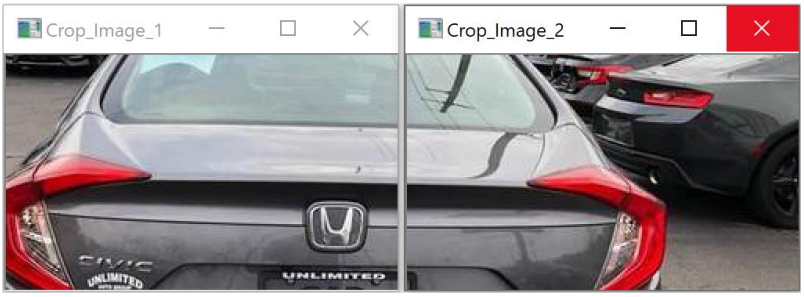
ARAÇ RENK ALGILAMA

Uygulamaya girdi olarak verilen araç görüntüsü üzerinde belirlediğimiz (y1:y2, x1:x2) koordinat sınırları içerisinde 2 farklı görüntü kırpılır. Şekil 3.1' de kırpılan görüntü üzerinde BGR değerleri bulunur. Blue için 0, Green için 1 ve Red için 2 değeri kullanılır. Daha sonra bu değerlerin 1/2 ortalaması alınarak görüntü içerisindeki baskın renk değeri olarak değerlendirilir. Yeni değerler bizim araç rengi için tahmin ettiğimiz değerler olur. Belirlediğimiz renklerin örneğin; siyah, beyaz, kırmızı, mavi gibi renklerin alt ve üst değerleri tarafımızca tayin edilir. Ortalama olarak bulduğumuz BGR değerleri bu aralıklara if şart algoritması kullanılarak bir sonuç tahmininde bulunulur.



Şekil 3.1: Renk algılama da kırpılan bölgeler örneği

Kullanılan bu yöntem temel renklerde doğru sonuçlar verse de genel olarak handikapları olan bir yöntemdir. Örneğin Şekil 3.1' de gösterilen güneşli bir havada füme rengine sahip aracın görüntüsü çekildiğinde görüntü üzerinde güneşin etkisi ile oluşan parlaklık bu aracın görüntüsünü doğru olarak bulmamızda sistemimizi yanıltma etkisi mevcuttur.



Şekil 3.2: Güneş vb. etkenlerin renk algılamadaki olumsuz etkileri

Bu çalışmanın temel amacı marka/model tespiti olduğu için renk tanıma üzerine sade bir kurgu düşünülmüştür. Renk tanıma için Derin öğrenme yöntemleri kullanılarak daha iyi sonuçlar elde edilir.

DÖRDÜNCÜ BÖLÜM
PLAKA TANIMA SİSTEMİ

Plaka tanıma sistemi bu çalışma için temel olarak 6 adımdan oluşmaktadır.

- Görüntü almak için kullanılan bir cihaz yardımıyla aracın görüntüsü önden ya da arkadan plaka bölgesi görünecek şekilde görüntü elde edilir
- Elde edilen bu renkli görüntü, tümüyle gri renk olacak şekilde çevrilir.
- Yeni oluşan görüntü üzerinde bir filtre yardımıyla yumuşatma işlemi yapılır.
- Oluşan görüntü üzerinde bir kenar belirleme algoritması kullanılarak görüntü üzerindeki kenar tepe noktaları belirlenir
- Belirlenen tepe noktaları arasında gezinme yapılarak plaka bölgesi belirlenir.
- Plaka bölgesi elde edildikten sonra görüntü üzerinde karakter tanıma yöntemleri kullanılarak harf ve rakamlar belirlenir.

Yukarıda bahsedilen adımları biraz daha açıklayacak olursak, elde edilen aracın görüntüsü RGB değerleri taşıyan renkli bir görüntüden oluşmaktadır. Görüntü işleme üzerinde çalışıldığında problemin temel olarak ne olduğu belirlenir. Yani problem eğer bir meyve sınıflandırma üzerine ise burada renklerin önemi en temel esastır. Örneğin elma için yeşil elma, kırmızı elma veya sarı elma olup olmadığını belirleme noktasında rengin önemi birinci sıradadır.

Fakat bizim problemimizde hedefin aracın plaka bölgesi olduğu için renkli bir görüntüye ihtiyaç duyulmamaktadır. Buradaki temel amaç alınan görüntünün renkli olması üzerinde plaka bölgesinin tespiti ya da bir noktanın tespiti için çok farklı renkler üzerinde çalışma yapılmasının önüne geçmektir. Bu gibi handikaplı durumların önüne geçilmesi için en temel işlem gri çevirme yöntemine başvurmaktır. Ayrıca sistemin daha hızlı çalışabilmesi için renkli görüntülerin handikap oluşturacağından dolayı RGB görüntüsü tercih edilmemelidir.

Daha doğru ve kararlı bir sonuç alabilmek için renkli olan araç görüntüsü alındıktan sonra bir komut sayesinde tümüyle griye çevirilir (Sathya et al., 2017). Griye çevirilen görüntü üzerinde var olan gürültülü noktalar bizim için olumsuz bir durum teşkil edebilir. Bu sebepten ötürü görüntü üzerinde bir takım filtre veya filtreler

yardımıyla görüntü daha basitleştirilerek algoritmanın daha doğru ve hızlı sonuç elde edebilmesi için görüntü üzerinde yumuşatma diye tabir edilen adım gerçekleştirilir. Bu çalışmada yumuşatma işlemi için Bilateral Filtering uygulanmıştır.

Bilateral Filter, görüntüler için doğrusal olmayan gürültüleri azaltmak ve kenarları koruyucu belirgin kılan bir yumuşatma filtresi olarak kullanılır. Her pikseliğin yoğunluğunu, yakındaki piksellerden hesapladığı ortalama yoğunluk değerleri ile değiştirir. Bu filtre sayesinde görüntü üzerinde yumuşatma işlemi gerçekleşmiş olurken aynı zamanda kenar bölgelerinde daha net keskin hatlar oluşmasını sağlar. Bilateral filtresinin bir sonraki adıma katkısı da tam olarak burada belirli hale gelir. Çünkü oluşturulan yeni görüntü üzerinde nesnelerin belirlenebilmesi için kenarlarının belirlenmesi gerekir. Bu çalışmamızda da plaka bölgesinin tespiti için kenarlar önemlidir.

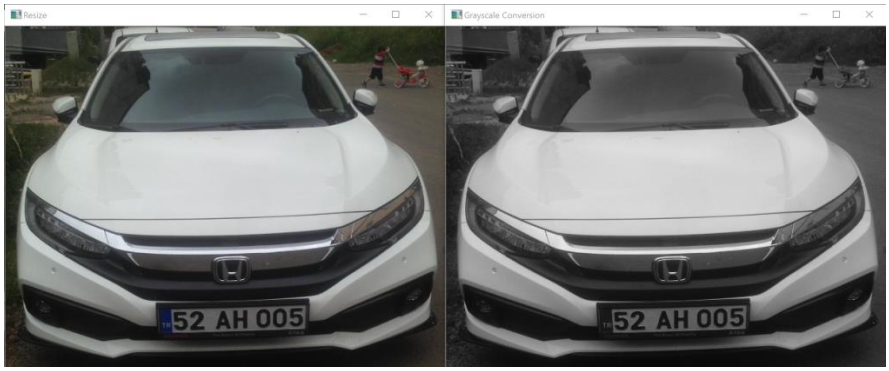
Bu bahsedilen adım için kenar belirleme algoritmaları olarak literatürde Sobel, Prewitt, Roberts, Laplacian ve Canny algoritmaları kullanılır. Kenar belirleyici algoritmalar problemin durumuna göre başarıda farklılık gösterir. Bu çalışmada ise Canny Edge Detection algoritması tercih edilmiştir. Bahsedilen bu adımların literatürdeki karşılıkları için ayrıntılı olarak bahsetmek gerekirse;

4.1. Gri Tonlamalı Dönüşüm (Grayscale Conversion)

Genel olarak piksel başına 8 bit ayrılır. 256 renk için her rengin tonu kadar gri ton bulunduran gri ölçektir. Yani 256 tane farklı gri ton demektir. Orijinal araç görüntüsü BGR olarak gelir. 3 ana renkten oluşan görüntü üzerinde daha hızlı işlemler yapabilmek için resim griye dönüştürülür. Örneğin renkli bir pikselin renk değerinin BGR (140,55,210) olduğunu varsayalım. Burada 3 sayıda farklı olduğu için renkli resim olarak ifade edilir. Eğer B, G ve R değerleri eşit olursa 0 ile 255 aralığında bir gri renk değerine karşılık gelir (Chowdhury et al., 2019). Buradan yola çıkarak resmin tamamını griye çevirmek için B, G ve R değerleri toplanır ve 3 bölünerek ortalaması alınır. Örneğin BGR (100,50,150) değeri BGR (100,100,100) şeklinde gri bir resim olacaktır. Her piksel için uygulanacak formül;

$$x = \frac{B_{ij}+G_{ij}+R_{ij}}{3} \qquad (4.1.1.1)$$

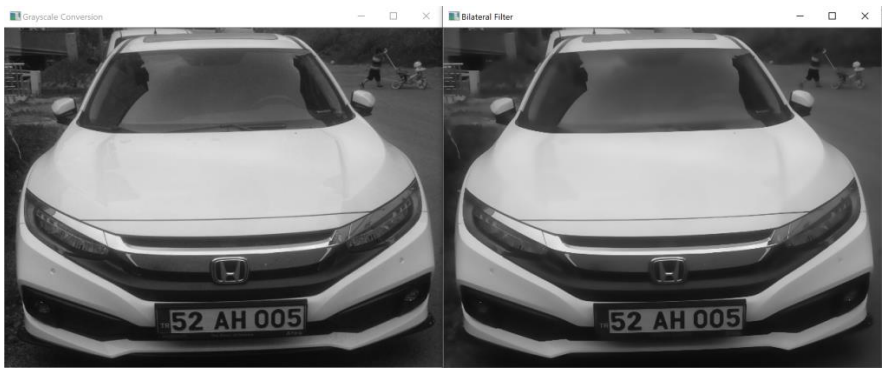
Her P[i,j] pikseli için formülden bulunan BGR (x, x, x) değeri gri değerimizdir.



Şekil 4.1.1: Görüntünün griye çevirilmesi

4.2. İki Taraflı (Bilateral) Filter

1998 yılında Tomasi ve Manduchi tarafından ortaya çıkarılmıştır (Tomasi & Manduchi, 1998). Methotdun çalışma prensibi yakın piksellerin geometrik olarak uzaklığı ve gri seviye benzerlikleri üzerine kuruludur. Yerel bir komşulukta piksel değerlerinin ağırlıklı toplamını almaktadır. Bu ağırlık hem Öklid mesafesine hem de piksel değerinin yoğunluk mesafesine göre değişebilmektedir. Bu methodu diğer methodlardan ayıran en önemli farkı ise görüntü üzerindeki gürültülerin azalmasının yanı sıra görüntü üzerindeki kenar noktalarının korunarak daha belirgin halde kalmasını sağlanmasıdır (Akar et al., 2015). Bu şekilde kenar noktaları belirgin kalması plaka bölgesinin tespiti için çok önemli bir adımdır.



Şekil 4.2.1: İki taraflı filtrenin uygulanması

4.3. Canny Kenar Belirleme Algoritması

Kenar bulmak için gayet başarılı bir algoritmadır. İlk işlem görüntü üzerindeki gürültü bir sigma değeri ile Gaussian çekirdekle üretilen değer ile konvolüsyonu

alınarak azaltılmasıdır (Isra & Gokulanathan, 2017). Daha sonra gradyent operatörü uygulanılarak kenar gradyentin büyüklüğü ve yönü hesaplanır. Kenarları inceltilir ve görüntüye ikili eşikleme uygulanarak istenmeyen gürültü ayrıntılarından temizlenir (Turan, 2017). Parametre olarak iki sayı eşikleri(threshold) belirtir. İkinci argümana düşük eşik değeri (low threshold) ve üçüncü argümana yüksek eşik değeri (high threshold) adı verilir. Eğer gradient değeri yüksek eşik değerinden fazla olursa keskin bir kenar olarak işaretlenir. Canny kenar detektörü bu noktadan kenarı takip etmeye başlar ve gradient değeri düşük eşik değerinin altına inene kadar işlemi devam eder. Canny kenar belirleme algoritması 4 aşamadan oluşmaktadır (Akar E. O., 1393).

4.3.1. Bulanıklaştırma (Smoothing)

Orijinal görüntüye Gaussian filtresi ile bulanıklaştırılır. K[i,j] Orijinal görüntü, R[i,j;σ] Gaussian bulanıklaştırıcı filtre ve σ ; Gaussian filtresinin standart sapması (yumuşatma derecesi) olmak üzere, orijinal K[i,j] görüntüsü ve R[i,j;σ] filtresinin konvolüsyonu sonucu elde edilen yumuşatılmış görüntü Y[i,j] ile ifade edilmektedir (Turan, 2017).

$$Y[i,j] = R[i,j;\sigma] * K[i,j] \quad (4.3.1.1)$$

4.3.2. Gradyanın Hesaplanması

(2.9.3.1.1)' de verilen Y[i,j]'nin kısmi türevleri elde edilir.

$$L[i,j] \approx (Y[i,j+1] - Y[i,j] + Y[i+1,j+1] - Y[i+1,j]) / 2 \quad (4.3.2.1)$$

$$W[i,j] \approx (Y[i,j] - Y[i+1,j] + Y[i,j+1] - Y[i+1,j+1]) / 2 \quad (4.3.2.2)$$

Şeklindedir. x ve y kısmi türevleri 2x2'lik kare matris üzerindeki sonlu farkların ortalaması alınarak hesaplanır. Buna göre gradyentin büyüklük değeri:

$$D(i,j) = \sqrt{L[i,j]\,2 + W[i,j]\,2} \quad (4.3.2.3)$$

ve açısı: $W[i,j] = \arctan(W[i,j], L[i,j])$ (4.3.2.4)

olur (Kısa, 2014).

4.3.3. Maksimum Olmayan Noktaların Bastırılması

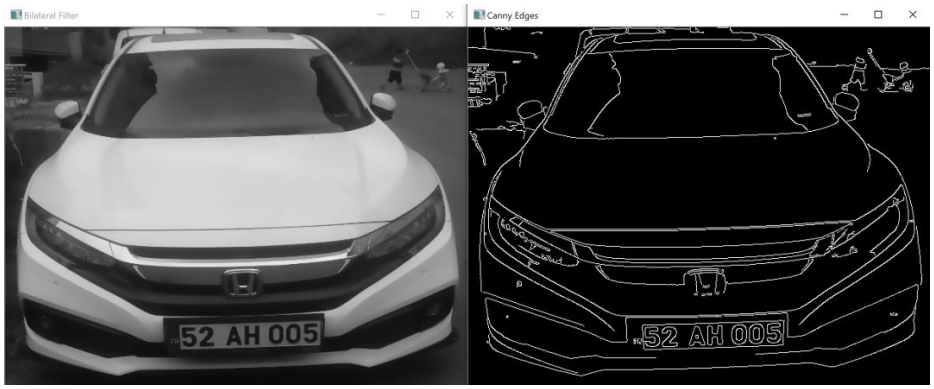
Gradyent algoritmasında, giriş görüntüsünün büyüklük değerinin gradyenti alınarak kenar pikselleri belirlemek mümkündür. Fakat Canny metodu karmaşık bir yaklaşıma sahiptir. Bu yaklaşım kenar piksellerinde meydana gelen çizgi hatlarını inceltme adımıdır. Bu adım sonrası yeni görüntü F[i,j] görüntüsüdür (Arslan, 2011).

$$F[i, j] = nms(D[i, j], \xi[i, j]) \text{ (Kısa, 2014)} \quad (4.3.3.1)$$

4.3.4. Eşikleme

Görüntü üzerinde çift eşikleme methodu ile kenar pikseller tespit edilir. Maksimum olmayan noktaları bastırılmış olan F[i,j] görüntüsünde ilk adımda smoothing işlemi yapıldığı halde resimde bulunan gürültüler nedeniyle kenar noktaların bazılarında hatalar oluşması beklenebilir. Bu hatalı noktaları azaltmak için F[i,j]' ye bir eşik değeri uygulanır ve eşik değerinin altında kalan tüm noktalar 0 yapılmaktadır (Arslan, 2011). Burada eşikleme işlemi yapıldıktan sonra elde edilen yeni görüntü kenarları belirlenmiş E[i,j] görüntüsüdür. Fakat burada dikkat edilmesi gereken husus uygun bir eşik değeri belirlemektir.

Bu eşik değeri genellikle deneme yanılma yöntemi ile seçilir. Bu handikaplı yöntemin üstesinden çift eşikleme yöntemi kullanılarak gelinir. Bu yöntem sayesinde F[i,j] görüntüsüne C1 ve C2 olmak üzere iki eşik değeri uygulanır. Uygulanan bu adım sonrasında C1 [i,j] ve C2 [i,j] eşiklenmiş görüntüleri oluşur. C2 görüntüsü üzerindeki kenar piksellerde oluşturulan hat üzerinde açıklıklar oluşur. Ancak dışarı kalan istenmeyen kenarların sayısı dikkate alınacak kadar değildir. Sadece açıklıkların kapatılması için C1 görüntüsü kullanılarak görüntü üzerinde optimum düzeltme sağlanmış olur (Turan, 2017).



Şekil 4.3.4.1: Canny Kenar belirleme algoritmasının uygulanması

4.4. Görüntü Üzerinde Plaka Bölgesinin Tespiti

Canny kenar dedektörü sayesinde oluşturulan yeni görüntü üzerinde görüleceği üzere plaka bölgesi dikdörtgen şekline benzer bir kenar bölgesi şeklindedir. Plaka bölgesini

bulabilmek için görüntüdeki konturları bulmamız gereklidir. Kontur, bir nesnenin ana hatlarını veya şeklini ifade eder.

Bir görüntü üzerinde konturları tespit edebilmek için OpenCV komutu olan cv2.findContuors komutu kullanılır. Bu komutun kullanımı için üç parametre gerekir. Birinci parametre, içinde kenarlarını bulmak istediğimiz görüntünün kopyalanmasıdır. İkinci parametre ise cv2.RETR_TREE komutudur. Bu komut OpenCV' ye konturlar arasındaki hiyerarşik ilişkiyi hesaplamasını söyler. Son parametre ise cv2.CHAIN_APPROX_SIMPLE komutu ile yatay, dikey ve diyagonal bölümleri sıkıştırır ve yalnızca uç noktaları bırakır.

Örneğin bir dikdörtgensel kontur 4 nokta ile kodlanır. Tespit edilen konturlar bir dizide saklanır. Görüntü üzerindeki hangi konturun plaka bölgesine ait olduğunu tespit edebilmek için kontur alanlarını cv2.contourArea komutu kullanılarak hesaplanır. Konturlar en büyükten en küçüğe sıralanır. Belirlediğimiz 10 sayısı kadar en büyük kontur değerleri tutulur. Ardından konturu cv2.arcLength ve cv2.approxPolyDp kullanarak yaklaştırıyoruz.

Bu yöntemler bir konturun çokgen eğrilerini tahmin etmek için kullanılır. Bir kontura yaklaşmak için yaklaşıklık hassasiyeti seviyesi sağlanır. Bu durumda kontur çevresinin %1,8' ini kullanıyoruz. Hassasiyet uygulanacak projede değişiklik gösterir. En iyi sonuç için deneme yanılma yöntemine başvurulur. Plaka bölgesinin dikdörtgen görüntüsü olduğunu biliyoruz. Dolayısıyla dört köşesi olduğu bilinen kontur noktası için bir "if" şartı kullanılarak kontur noktası 4 olan muhtemelen plaka bölgesi olarak seçilir. Seçilen bölge çizgi ile renklendirilir. Daha sonra plaka bölgesi olarak seçilen bölge görüntüsü kırpılmak üzere maskelenir. Maskelenen görüntü yeni bir resim olarak kaydedilir.

Şekil 4.4.1: Plaka bölgesinin tespiti

Maskelenmiş görüntü üzerinde x ve y noktalarının alt ve üst nokta sınırları içerisinde görüntü kırpılır. Kırpılan görüntü artık plakanın kendisidir.

Şekil 4.4.2: Tespit edilen plaka bölgesinin kırpılması

Karakter tanıma için kullanılan "pytesseract" komutu yardımıyla plaka üzerinde bulunan harf ve rakamlar tanınarak tahmin edilir (Dias et al., 2019). Tanınan plaka ekrana yazdırılır.

https://www.sahibinden.com/(*Sahibinden*, n.d.) web sitesinden test için alınan 16 adet araç görüntüsü kullanılmadan önce Kişisel Verileri Koruma Kanunu kapsamında her bir araç görüntüsü üzerinde bulunan sahibinden.com web sitesine ait watermark, photoshop yardımıyla görüntü üzerinden kaldırıldı. Bu işlemin ardından aynı kanun kapsamında her bir araç görüntüsü üzerindeki plakalar gerçeği yansıtmayan plakalar olarak düzenlenerek değiştirilmiştir.

Bu çalışma da kullanılan test görüntüleri için çözünürlük değerleri yaklaşık olarak en az 500 x 500 piksel ölçülerinde olacak şekilde test edilmiştir. Görüntünün boyutları düşük piksel üzerinde test edildiğinde renk tespiti, plaka bölgesinin tespiti veya karakter okuma işleminde başarısızlıkla sonuçlandığı görülmüştür. Belirlenen ölçüler baz alınarak gerçekleştirilen test sonuçları Tablo 4.4.1' de gösterildiği gibidir. 16 adet test görüntüsü için %87,5 doğruluk oranı ile 14 adet görüntüde başarılı sonuç

elde edilirken 2 adet görüntüde çözünürlüğün düşük olması sebebiyle %12,5 başarısızlık oranı ile test sonuçlanmıştır.

Tablo 4.4.1: Plaka ve renk test sonuçları

Örnekler	**Sonuçlar**
Doğru Renk ve Doğru Plaka	14
Doğru Renk ve Hatalı Plaka Okuma	1
Yanlış Renk ve Plaka Tespit Edilemedi	1
	16

BEŞİNCİ BÖLÜM

VERİ TABANI ARAÇ KAYIT ESLEŞTİRME

Marka/model, plaka ve renk bilgisi alınan aracın bilgileri, bu çalışma kapsamında gerçekliği olmayan veriler ile oluşturulan veri tabanı içerisindeki kayıtlar ile karşılaştırılmıştır. Kayıtlar içerisinde aracın marka/model, plaka ve renk bilgisi eşleştiğinde ekrana eşleşme başarılı mesajı yayınlaması, eşleşme de herhangi bir bilginin eşleşmemesi durumunda eşleşme başarısız mesajı vermesini tanımlanmıştır. Bu bilgiler ve denenecek olan araç görüntüleri sözde doğruluk ile bu çalışma kapsamında rastgele belirlenmiştir. Başarılı ve Başarısız Eşleşme sonuçlarına ait birkaç adet örnek görüntülere aşağıdaki bölümlerde yer verilmiştir.

Tablo 5.1: Sanal_Test_Veritabanı

id	marka_model	plaka	renk
1	2012_2014_Ford Focus Ön	77AFL452	Kırmızı
2	2012_2014_Ford Focus Ön	71TAT711	Beyaz
3	2012_2014_Ford Focus Ön	66BET601	Kırmızı
4	2012_2014_Ford Focus Ön	31SK333	Siyah
5	2012_2014_Ford Focus Ön	88UU8067	Beyaz
6	2016_2019_Honda Civic Ön	35AEE539	Siyah
7	2016_2019_Honda Civic Ön	52ABC123	Siyah
8	2016_2019_Honda Civic Ön	70CC700	Siyah
9	2012_2014_Ford Focus Ön	61KAK052	Beyaz
10	2012_2014_Ford Focus Ön	06BBT601	Beyaz
11	2012_2014_Ford Focus Ön	52CAT777	Beyaz
12	2012_2014_Ford Focus Ön	97AGD900	Beyaz
13	2012_2014_Ford Focus Arka	95ADD319	Beyaz
14	2012_2014_Ford Focus Ön	77FAL555	Kırmızı

Şekil 5.1: Test_1

```
Araç Marka Model: 2012_2014_Ford Focus Ön
Aracın Rengi    : Kırmızı
Aracın Plakası  : 77AFL452
Araç Marka Model, Plaka ve Renk Bilgisi Eşleşti...
```

Şekil 5.2: Başarılı eşleşme sonucu bilgisi_1

Şekil 5.3: Test_2

```
Araç Marka Model: 2012_2014_Ford Focus Ön
Aracın Rengi    : Beyaz
Aracın Plakası  : 71TAT711
Araç Marka Model, Plaka ve Renk Bilgisi Eşleşti...
```

Şekil 5.4: Başarılı eşleşme sonucu bilgisi_2

Şekil 5.5: Test_3

```
Araç Marka Model: 2012_2014_Ford Focus Ön
Aracın Rengi    : Kırmızı
Aracın Plakası  : 66BET601
Araç Marka Model, Plaka ve Renk Bilgisi Eşleşti...
```

Şekil 5.6: Başarılı eşleşme sonucu bilgisi_3

Şekil 5.7: Test_4

```
Araç Marka Model: 2016_2019_Honda Civic Ön
Aracın Rengi    : Siyah
Aracın Plakası  : 35AEE539
Araç Marka Model, Plaka ve Renk Bilgisi Eşleşti...
```

Şekil 5.8: Başarılı eşleşme sonucu bilgisi_4

Şekil 5.9: Test_5

```
Araç Marka Model: 2016_2019_Honda Civic Ön
Aracın Rengi    : Siyah
Aracın Plakası  : 70CC700
Araç Marka Model, Plaka ve Renk Bilgisi Eşleşti...
```

Şekil 5.10: Başarılı eşleşme sonucu bilgisi_5

Şekil 5.11: Test_6

```
Araç Marka Model: 2016_2019_Honda Civic Ön
Aracın Rengi    : Siyah
Aracın Plakası  : 93ABM933
Eşleşme Başarısız..!
```

Şekil 5.12: Başarısız eşleşme sonucu bilgisi_1

Şekil 5.13: Test_7

```
Araç Marka Model: 2012_2014_Ford Focus Ön
Aracın Rengi    : Beyaz
Aracın Plakası  : 96KL684
Eşleşme Başarısız..!
```

Şekil 5.14: Başarısız eşleşme sonucu bilgisi_2

Şekil 5.15: Test_8

```
Araç Marka Model: 2012_2014_Ford Focus Ön
Aracın Rengi    : Beyaz
Aracın Plakası  : 77CAT712
Eşleşme Başarısız..!
```

Şekil5.16: Başarısız eşleşme sonucu bilgisi_3

Şekil 5.17: Test_9

```
Araç Marka Model: 2012_2014_Ford Focus Ön
Aracın Rengi    : Kırmızı
Aracın Plakası  : 77FFL552
Eşleşme Başarısız..!
```

Şekil 5.18: Başarısız eşleşme sonucu bilgisi_4

SONUÇ

Bu çalışmanın sonucunda sisteme girdi olarak verilen araç görüntüleri üzerinde yapılan çalışmalar, aracın marka/model bilgisi üzerinde %99' a yakın doğru sonuç ürettiği plaka okuma işleminde ise belirli piksel ölçüleri baz alınarak test edilen görüntüler üzerinde %87,5 başarı oranın yanında, düşük piksel ölçüleri sebebiyle oluşan %12,5 başarısızlık oranı gözlemlenmiştir. Marka model bilgisinin yüksek doğruluk oranına sahip olmasının en temel etkeni, sistemin derin öğrenme kullanılarak eğitilip test edilmesidir. Günümüzde kullanılan en popüler yöntemlerden birisi de artık derin öğrenme olduğu gerçeğini kendi çalışmamızda da test ederek gözlemlenmiş oluyoruz. Çalışmadaki bir diğer eklenti ise aracın özelliklerinden renk tanıma işlemidir. İşlem düz mantık ile herhangi bir yapay sinir ağı vs. gibi öğrenme, eğitme, test etme işlemine tabi olmaksızın araç görüntüsü üzerinden belirli koordinatlar verilerek alınan görüntü parçaları üzerindeki piksel sayılarının toplamı ve ortalamasının bulunması şeklinde tasarlanmıştır. Basit ve sade bir yöntem olan bu işlem için en büyük handikap güneşli havalardaki ışığın yansıması örneğidir. İlgili örneğe çalışmanın renk bulma bölümünde yer verilmiştir. Son olarak aracın özellik çıkarımlarından plaka tanıma ile çalışma sonlandırılmıştır. Plaka tanıma için de aynı şekilde derin öğrenme kullanılmamış olup belirli morfolojik işlemler adımı uygulanarak plaka bölgesi tespit edilmiştir. Tespit edilen görüntü üzerinde plaka tespiti için karakter tanımaya yarayan bir fonksiyon kullanılmıştır. Bu çalışmaya ait veri setlerine ve kod bloğuna erişim için aşağıdaki Github linki üzerinden erişim sağlayabilirsiniz.

https://github.com/burakaggul/Vehicle-brand-model-recognition-with-deep-learning-using-keras

KAYNAKÇA

Abdulkadir, Ş. & Yüksek, A. G. (2017). Stacked Autoencoder Method for Fabric Defect Detection. *Cumhuriyet Science Journal*, *38*(2), 342–342. https://doi.org/10.17776/cumuscij.300261

autoscout24. (n.d.). https://www.autoscout24.com.tr/

Carlson, K. J. (2019). The military application of artificial intelligence. In *Bowie State University* (pp. 1–21).

Chowdhury, D., Mandal, S., Das, D., Banerjee, S., Shome, S., & Choudhary, D. (2019). An adaptive technique for computer vision based vehicles license plate detection system. *2019 International Conference on Opto-Electronics and Applied Optics, Optronix 2019*. https://doi.org/10.1109/OPTRONIX.2019.8862406

Chung, H., Lee, S. J., & Park, J. G. (2016). Deep neural network using trainable activation functions. *Proceedings of the International Joint Conference on Neural Networks*, *2016-October*(l), 348–352. https://doi.org/10.1109/IJCNN.2016.7727219

Craigslist.org. (n.d.). https://cfl.craigslist.org/

Dias, C., Jagetiya, A., & Chaurasia, S. (2019). Anonymous vehicle detection for secure campuses: A framework for license plate recognition using deep learning. *2019 2nd International Conference on Intelligent Communication and Computational Techniques, ICCT 2019*, 79–82. https://doi.org/10.1109/ICCT46177.2019.8969068

Huval, B., Wang, T., Tandon, S., Kiske, J., Song, W., Pazhayampallil, J., Andriluka, M., Rajpurkar, P., Migimatsu, T., Cheng-Yue, R., Mujica, F., Coates, A., & Ng, A. Y. (2015). *An Empirical Evaluation of Deep Learning on Highway Driving*. 1–7. http://arxiv.org/abs/1504.01716

Isra, A. R. & Gokulanathan, A. (2017). Vertical-Edge-Based Car-License-Plate Detection Method. *IOSR Journal of Electrical and Electronics Engineering*, *12*(01), 01–06. https://doi.org/10.9790/1676-1201020106

Kaya, U., Yılmaz, A. & Dikmen, Y. (2019). Sağlık Alanında Kullanılan Derin Öğrenme Yöntemleri. *European Journal of Science and Technology*, *16*, 792–808. https://doi.org/10.31590/ejosat.573248

Lau, M. M. & Lim, K. H. (2019). Review of adaptive activation function in deep neural network. *2018 IEEE EMBS Conference on Biomedical Engineering and Sciences, IECBES 2018 - Proceedings*, 686–690. https://doi.org/10.1109/IECBES.2018.08626714

Miao, Y., Gowayyed, M. & Metze, F. (2016). EESEN: End-to-end speech recognition using deep RNN models and WFST-based decoding. *2015 IEEE Workshop on Automatic Speech Recognition and Understanding, ASRU 2015 - Proceedings*, 167–174. https://doi.org/10.1109/ASRU.2015.7404790

Öztürk, K. & Şahin, M. E. (2018). *A General View of Artificial Neural Networks and Artificial Intelligence*. *6*(2), 25–36. http://www.sloi.org/sloi-name-of-this-article

Ray, S. (2011). An Overview of the Tesseract OCR Engine. *Communications in Computer and Information Science*, *167 CCIS*(PART 2), 686–691. https://doi.org/10.1007/978-3-642-22027-2_57

Sahibinden. (n.d.). https://www.sahibinden.com/

Sathya, K. B., Vaidehi, V. & Kavitha, G. (2017). Vehicle License Plate Recognition (VLPR). *Proceedings - TIMA 2017: 9th International Conference on Trends in Industrial Measurement and Automation*. https://doi.org/10.1109/TIMA.2017.8064786

Şeker, A, Diri, B. & Balık, H. (2017). Derin Öğrenme Yöntemleri ve Uygulamaları Hakkında Bir İnceleme. *Gazi Journal of Engineering Sciences*, *3*(3), 47–64. http://dergipark.gov.tr/gmbd/issue/31064/372661

Şeker, Abdulkadir. (2017). *DERİN ÖĞRENME YÖNTEMLERİ VE UYGULAMALARI HAKKINDA BİR İNCELEME ABDULKADİR*.

Sharma, O. (2019). A New Activation Function for Deep Neural Network. *Proceedings of the International Conference on Machine Learning, Big Data, Cloud and Parallel Computing: Trends, Prespectives and Prospects, COMITCon 2019*, 84–86.

https://doi.org/10.1109/COMITCon.2019.8862253

Sharma V. Avinash. (2017). Activation Functions in Neural Networks. *Medium*, *4*(12), 1–10.

Tan, H. H. & Lim, K. H. (2019). Vanishing Gradient Mitigation with Deep Learning Neural Network Optimization. *2019 7th International Conference on Smart Computing and Communications, ICSCC 2019*, 7–10. https://doi.org/10.1109/ICSCC.2019.8843652

Tan, Z. (2019). *DERİN ÖĞRENME YARDIMIYLA ARAÇ SINIFLANDIRMA*. https://doi.org/10.1017/CBO9781107415324.004

Yağcı, C., Gökçe, İ., Bozüyük, T. & Akar, G. (2005). *Yapay Zeka Teknolojilerinin Endüstriyel Uygulamaları*. 67.

Yi. (2018). *Groundbreaking Activation Functions*. https://codeodysseys.com/posts/activation-functions/

Yıldız, S., Özgür, E., Bilal, N., Yıldız, S., Özgür, E. & Bilal, N. (2019). *Yapay Zeka Tabanlı Yüz Tanı ma Sisteminin Geliştirilmesi Ve Optimizasyonu Development and Optimization of Artificial Intelligence Based Face Recognition System*.

EK-1 Marka/Model/Renk/Plaka Tespit

###Modelin Eğitilmesi

```
import numpy as np
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras import optimizers
classifier = Sequential()
classifier.add(Conv2D(32, (3, 3), input_shape=(600, 450, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Conv2D(64, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Conv2D(128, (3, 3), activation='relu'))
classifier.add(MaxPooling2D(pool_size=(2, 2)))
classifier.add(Flatten())
classifier.add(Dense(units=128, activation='relu'))
classifier.add(Dropout(0.5))
classifier.add(Dense(units=4, activation='softmax'))
classifier.compile(
   optimizer='adam', loss='categorical_crossentropy', metrics=['accuracy'])
train_datagen = ImageDataGenerator(rescale=1./255,
                                   shear_range=0.2,
                                   zoom_range=0.2,
                                   horizontal_flip=True)
```

```
test_datagen = ImageDataGenerator(rescale=1./255)
training_set = train_datagen.flow_from_directory('training_set',
                                                 target_size=(600, 450),
                                                 batch_size=32,
                                                 class_mode='categorical')
test_set = test_datagen.flow_from_directory('test_set',
                                            target_size=(600, 450),
                                            batch_size=32,
                                            class_mode='categorical')
history = classifier.fit_generator(training_set,
                                   steps_per_epoch=10000/32,
                                   epochs=40,
                                   validation_data=test_set,
                                   validation_steps=1000/32)
loss = history.history['loss']
val_loss = history.history['val_loss']
epochs = range(len(loss))
plt.figure()
plt.plot(epochs, loss, 'bo', label='Training loss')
plt.plot(epochs, val_loss, 'b', label='Validation loss')
plt.title('Training and validation loss')
plt.legend()
plt.show()
accuracy = history.history['accuracy']
val_accuracy = history.history['val_accuracy']
epochs = range(len(accuracy))
plt.figure()
plt.plot(epochs, accuracy, 'bo', label='Training accuracy')
plt.plot(epochs, val_accuracy, 'b', label='Validation accuracy')
plt.title('Training and validation accuracy')
plt.legend()
plt.show()
```

```
classifier.save('model_600_450_32_categorical.h5')
```

###Eğitilen Modelin Yüklenmesi

```
import cv2
from PIL import Image
import numpy as np
import imutils
import pytesseract
import re
from keras.applications.vgg16 import preprocess_input
from keras.preprocessing import image
import matplotlib.pyplot as plt
from keras.preprocessing.image import ImageDataGenerator
from keras.models import Sequential
from keras.layers import Conv2D
from keras.layers import MaxPooling2D
from keras.layers import Flatten
from keras.layers import Dense
from keras.layers import Dropout
from keras import optimizers
from keras.models import load_model
import sqlite3
classifier = load_model('model_600_450_32_categorical.h5')
img_path = 'C:/Users/MacBookPro/test.jpg'
image_color = cv2.imread('C:/Users/MacBookPro/test.jpg',cv2.IMREAD_COLOR)
img_plaka = cv2.imread('C:/Users/MacBookPro/test.jpg',cv2.IMREAD_COLOR)
img = image.load_img(img_path, target_size=(600, 450))
x = image.img_to_array(img)
x = np.expand_dims(x, axis=0)
array = classifier.predict(x)
```

###Sınıflandırma

```
predicted_class_indices=np.argmax(array,axis=1)
if predicted_class_indices == 0:
```

```
    marka_m = "2012_2014_Ford Focus Ön"
    print("Araç Marka Model:",marka_m)
if predicted_class_indices == 1:
    marka_m = "2012_2014_Ford Focus Arka"
    print("Araç Marka Model:",marka_m)
if predicted_class_indices == 2:
    marka_m = "2016_2019_Honda Civic Ön"
    print("Araç Marka Model:",marka_m)
if predicted_class_indices == 3:
    marka_m = "2016_2019_Honda Civic Arka"
    print("Araç Marka Model:",marka_m)
```

###Renk Bulma

```
image_color = cv2.resize(image_color, (600,450))
cropped_1 = image_color[100:250, 100:350]
b_1 = int(cropped_1.item(100, 100, 0))
g_1 = int(cropped_1.item(100, 100, 1))
r_1 = int(cropped_1.item(100, 100, 2))
cropped_2 = image_color[100:250, 350:600]
b_2 = int(cropped_2.item(100, 100, 0))
g_2 = int(cropped_2.item(100, 100, 1))
r_2 = int(cropped_2.item(100, 100, 2))
cv2.imshow("Crop_Image_1", cropped_1)
cv2.imshow("Crop_Image_2", cropped_2)
ort_b = int((b_1)+(b_2))/2
ort_g = int((g_1)+(g_2))/2
ort_r = int((r_1)+(r_2))/2
if (180<=ort_b and ort_b<255 and 180<=ort_g and ort_g<255 and 170<=ort_r and
ort_r<255):
    arac_r = "Beyaz"
    print("Aracın Rengi    :",arac_r)
elif (110<ort_b and ort_b<150 and 110<ort_g and ort_g<150 and 110<ort_r and ort_r<150):
    arac_r = "Füme"
```

```
    print("Aracın Rengi    :",arac_r)
elif (150<ort_b and ort_b<180 and 150<ort_g and ort_g<180 and 150<ort_r and ort_r<170):
    arac_r = "Gri"
    print("Aracın Rengi    :",arac_r)
elif (130<ort_b and ort_b<255 and 25<ort_g and ort_g<150 and 0<ort_r and ort_r<150):
    arac_r = "Lacivert"
    print("Aracın Rengi    :",arac_r)
elif (0<ort_b and ort_b<150 and 0<ort_g and ort_g<150 and 100<ort_r and ort_r<255):
    arac_r = "Kırmızı"
    print("Aracın Rengi    :",arac_r)
elif (ort_b<=110 and ort_g<110 and ort_r<110):
    arac_r = "Siyah"
    print("Aracın Rengi    :",arac_r)
```

###Plaka Bulma

```
gray = cv2.cvtColor(img_plaka, cv2.COLOR_BGR2GRAY)
cv2.imshow("Grayscale Conversion", gray)
gray = cv2.bilateralFilter(gray, 11, 17, 17)
cv2.imshow("Bilateral Filter", gray)
edged = cv2.Canny(gray, 30, 200)
cv2.imshow("Canny Edges", edged)
cnts = cv2.findContours(edged.copy(), cv2.RETR_TREE, cv2.CHAIN_APPROX_SIMPLE)
cnts = imutils.grab_contours(cnts)
cnts = sorted(cnts, key = cv2.contourArea, reverse = True)[:10]
screenCnt = None
for c in cnts:
        peri = cv2.arcLength(c, True)
        approx = cv2.approxPolyDP(c, 0.02 * peri, True)
        if len(approx) == 4:
                screenCnt = approx
                break
if screenCnt is None:
        detected = 0
```

```
        print ("No contour detected")
else:
        detected = 1
if detected == 1:
        cv2.drawContours(img_plaka, [screenCnt], -1, (0, 255, 0), 2)
mask = np.zeros(gray.shape,np.uint8)
new_image = cv2.drawContours(mask,[screenCnt],0,255,-1,)
new_image = cv2.bitwise_and(img_plaka,img_plaka,mask=mask)
(x, y) = np.where(mask == 255)
(topx, topy) = (np.min(x), np.min(y))
(bottomx, bottomy) = (np.max(x), np.max(y))
Cropped = gray[topx:bottomx+1, topy:bottomy+1]
cv2.imshow("Final ",img_plaka)
cv2.imshow("Tespit Edilen Plaka",Cropped)
t_plaka = pytesseract.image_to_string(Cropped, config='--psm 11')
```

###Veri Tabanı Bağlantı

```
connectdb=sqlite3.connect('test.db')
im=connectdb.cursor()
plk = re.sub('\ |\?|\.|\!|\/|\\|\;|\:|\‘|\(|\)|\-|\{|\}|\]|\[|\&|\,|\’|\||\»|\¢|\*|\§|\°|',",t_plaka)
print("Aracın Plakası  :",plk)
im.execute("SELECT * FROM kayitli_araclar where marka_model='"+marka_m+"' and plaka='"+plk+"' and renk='"+arac_r+"'")
data = im.fetchone()
if data:
   print("Araç Marka Model, Plaka ve Renk Bilgisi Eşleşti...".format(data[0]))
else:
   print("Eşleşme Başarısız..!")
cv2.waitKey(0)
cv2.destroyAllWindows()
```

EK-2 Veri Artırma

```
import cv2
import numpy as np
from skimage import io
from skimage.transform import rotate, AffineTransform, warp
import matplotlib.pyplot as plt
import random
from skimage import img_as_ubyte
import os
from skimage.util import random_noise
img=cv2.imread(r"C:/Users/MacBookPro/Desktop/Drive/Honda/2012_2014_Ford_Focus_R
_07730.jpg")
img= cv2.cvtColor(img, cv2.COLOR_RGB2BGR)
plt.imshow(img)
plt.show()
height, width, dims= img.shape
print(height, width, dims)
gray= cv2.cvtColor(img, cv2.COLOR_RGB2GRAY)
print(gray.shape)
plt.subplot(121)
plt.imshow(img)
plt.title("Original Image")
plt.subplot(122)
plt.imshow(gray)
plt.title("Gray Image")
plt.show()
img= cv2.resize(img,(700,700))
img.shape
plt.subplot(131)
plt.title("Original Image")
plt.imshow(img)
hflipped_image= np.fliplr(img)
```

```
plt.subplot(132)
plt.title("Horizontally flipped")
plt.imshow(hflipped_image)
vflipped_image= np.flipud(img)
plt.subplot(133)
plt.title("Vertically flipped")
plt.imshow(vflipped_image)
plt.show()
r_image = rotate(img, angle=45)
r_image1 = rotate(img, angle=-45)
plt.subplot(131)
plt.imshow(img)
plt.title("Original Image")
plt.subplot(132)
plt.imshow(r_image)
plt.title("45 degree rotated image")
plt.subplot(133)
plt.imshow(r_image1)
plt.title("-45 degree rotated image")
plt.show()
transform = AffineTransform(translation=(-200,0))
warp_image = warp(img,transform, mode="wrap")
plt.subplot(1,2,1)
plt.title('original image')
plt.imshow(img)
plt.subplot(1,2,2)
plt.title('Wrap Shift')
plt.imshow(warp_image)
noisy_image= random_noise(img)
plt.subplot(1,2,1)
plt.title('original image')
plt.imshow(img)
```

```
plt.subplot(1,2,2)
plt.title('Image after adding noise')
plt.imshow(noisy_image)
blur_image= cv2.GaussianBlur(img, (11,11),0)
plt.subplot(1,2,1)
plt.title('original image')
plt.imshow(img)
plt.subplot(1,2,2)
plt.title('Blurry image')
plt.imshow(blur_image)
def anticlockwise_rotation(image):
    angle= random.randint(0,180)
    return rotate(image, angle)
def clockwise_rotation(image):
    angle= random.randint(0,180)
    return rotate(image, -angle)
def h_flip(image):
    return  np.fliplr(image)
def v_flip(image):
    return np.flipud(image)
def add_noise(image):
    return random_noise(image)
def blur_image(image):
    return cv2.GaussianBlur(img, (9,9),0)
def warp_shift(image):
    transform = AffineTransform(translation=(0,40))
    warp_image = warp(image, transform, mode="wrap")
    return warp_image
transformations = {'rotate anticlockwise': anticlockwise_rotation,
                   'rotate clockwise': clockwise_rotation,
                   'horizontal flip': h_flip,
                   'vertical flip': v_flip,
```

```
                'warp shift': warp_shift,
                'adding noise': add_noise,
                'blurring image':blur_image
               }
images_path="C:/Users/MacBookPro/Desktop/Drive/Honda"
augmented_path="C:/Users/MacBookPro/Desktop/Drive/Honda_Art"
images=[]
for im in os.listdir(images_path):
    images.append(os.path.join(images_path,im))
images_to_generate=40
i=1
while i<=images_to_generate:
    image=random.choice(images)
    original_image = io.imread(image)
    transformed_image=None
    n = 0
    transformation_count = random.randint(1, len(transformations))
    while n <= transformation_count:
        key = random.choice(list(transformations))
        transformed_image = transformations[key](original_image)
        n = n + 1
    new_image_path= "%s/augmented_image_%s.jpg" %(augmented_path, i)
    transformed_image = img_as_ubyte(transformed_image)
    transformed_image=cv2.cvtColor(transformed_image, cv2.COLOR_BGR2RGB)
    cv2.imwrite(new_image_path, transformed_image)
    i =i+1
```