T.C.
DOKUZ EYLÜL ÜNİVERSİTESİ
SOSYAL BİLİMLER ENSTİTÜSÜ
YÖNETİM BİLİŞİM SİSTEMLERİ ANABİLİM DALI
YÖNETİM BİLİŞİM SİSTEMLERİ PROGRAMI
YÜKSEK LİSANS TEZİ

DERİN ÖĞRENME TEKNİKLERİ İLE ANOMALİ İÇEREN METAL SOMUNLARIN HATA TESPİT VE SINIFLANDIRILMASI

Hasan GÖKKAYA

Danışman
Doç. Dr. Can AYDIN

İZMİR – 2020

YÜKSEK LİSANS
TEZ ONAY SAYFASI

Üniversite : Dokuz Eylül Üniversitesi

Enstitü : Sosyal Bilimler Enstitüsü

Adı ve Soyadı : Hasan GÖKKAYA

Öğrenci No : 2016800821

Tez Başlığı : Derin Öğrenme Teknikleri ile Anomali İçeren Metal Somunların Hata Tespit ve Sınıflandırılması

Savunma Tarihi : 16.01.2020

Danışmanı : Doç.Dr. Can AYDIN

JÜRİ ÜYELERİ

Ünvanı, Adı, Soyadı	Üniversitesi	İmza
Doç.Dr. Can AYDIN	- Dokuz Eylül Üniversitesi	.....
Prof.Dr.Vahap TECİM	- Dokuz Eylül Üniversitesi	.....
Doç:DrMurat KOMESLİ	- Yaşar Üniversitesi	.....

Hasan GÖKKAYA tarafından hazırlanmış ve sunulmuş olan bu tez savunmada başarılı bulunarak oy birliği(X) / oy çokluğu( ) ile kabul edilmiştir.

Prof. Dr. Erkan GÖKSU
Müdür V.

YEMİN METNİ

Yüksek Lisans Tezi olarak sunduğum “Derin Öğrenme Teknikleri ile Anomali İçeren Metal Somunların Hata Tespit ve Sınıflandırılması” adlı çalışmanın, tarafımdan, akademik kurallara ve etik değerlere uygun olarak yazıldığını ve yararlandığım eserlerin kaynakçada gösterilenlerden oluştuğunu, bunlara atıf yapılarak yararlanılmış olduğunu belirtir ve bunu onurumla doğrularım.

... /... / ……

Hasan GÖKKAYA

ÖZET

Yüksek Lisans Tezi

Derin Öğrenme Teknikleri ile Anomali İçeren Metal Somunların Hata Tespit ve Sınıflandırılması

Hasan GÖKKAYA

Dokuz Eylül Üniversitesi

Sosyal Bilimler Enstitüsü

Yönetim Bilişim Sistemleri Anabilim Dalı

Yönetim Bilişim Sistemleri Programı

Bilişim ve iletişim sistemlerinde meydana gelen hızlı değişimler ve gelişimler, teknolojinin temas ettiği her alanda ani bir paradigma değişimine neden olmuştur. Bu paradigma değişimi eş zamanlı olarak endüstriyel alanda da kendisini hissettirmiş, sektörün ihtiyaçları doğrultusunda karşılığını bulmuş ve sektörün içinde Endüstri 4.0 olarak vücut bulmuştur. 4'üncü Sanayi Devrimi olarak anılan bu paradigma, içerisinde yer alan nesnelerin interneti kavramıyla birlikte birbirleriyle anlık iletişim halinde olan modern otomasyon sistemlerini, üretim teknolojilerini ve nihayetinde de akıllı endüstriyel sistemleri hayata geçirmiştir. Ortaya çıkan akıllı iletişim ağında, paylaşılan verilerin raporlaştırılıp analiz edilmesi, maliyet-etkin bir biçimde yönetilebilmesi ve verimli iş modellerinin geliştirilmesi işletmeler için büyük önem arz etmektedir. Bu süreç, endüstriyel işletmelerin iş süreçlerini akıllı sistemler ile entegre ederek geleceğe uyumlu modern otomasyon ve üretim birimlerine dönüştürmesini zorunlu kılmaktadır.

Bilgisayarla görü uygulamaları, bu anlamda çağı yakalayan ve geleceğe uzanan çözümlerin başında gelmektedir. Otomasyon ve üretim sürecinden geçen işletmeler, makine öğrenmesi gibi teknikleri belirli operasyonlarına entegre etmeye başlamışlardır. Üretim süreci, endüstriyel işletmelerin hammadde kullanımı, kalite kontrol ve işletme verimliliği gibi ana başlıkları ile bir düğüm noktası oluşturduğu için özellikle bu entegrasyonda önem arz etmektedir. Bu

çalışma, endüstriyel alanda, hatalı ürünleri tespit etmek ve üretim verimliliğini optimize etmek için kullanılan akıllı bir yaklaşım önermektedir.

Bu çalışmanın amacı, vida ve somun üreten endüstriyel işletmelerde bilgisayarla görü ile insan faktörünün yerini alarak, üretim esnasında meydana gelen hatalı ürünleri tespit eden ve ayrıştırılmasını sağlayan, hata sebebinin belirlenmesine yardımcı olan, hata oranının düşürülmesine ve kalitenin artırılmasına katkıda bulunan, hammadde kullanımını optimize eden, üretim verimliliğini artıran ve sürecin maliyet-etkin bir şekilde işlemesini sağlayan derin öğrenmeye dayalı bir bilgisayarla görü uygulaması geliştirmektir. Bilgisayarla görü uygulaması tasarlanırken Python, OpenCV, Tensorflow Object Detection API, Tensorboard, GoogleColab, Visual Studio Code yazılım ve frameworkleri kullanılmıştır.

Anahtar Kelimeler: Anomali , Hata, Bilgisayarla Görü, Derin Öğrenme, Nesne Tespit, Sınıflandırma.

ABSTRACT

Master's Thesis

Fault Detection and Classification of Metal Nuts Containing Anomaly by Deep Learning Techniques

Hasan GÖKKAYA

Dokuz Eylul University

Graduate School of Social Sciences

Department of Management Information Systems

Management Information Systems Program

Rapid changes and developments in information and communication systems have caused a sudden paradigm shift in every field connected with technology. This paradigm shift made its presence felt simultaneously in the industrial field, has found its place in accordance with the needs of the sector and embodied as Industry 4.0 within the sector. This paradigm, known as the 4th Industrial Revolution, with the help of concept of the IoT, has implemented modern automation systems, production technologies and intelligent industrial systems, which are in instant communication with each other. In this smart communication network, reporting, analysis and cost-effective management of shared data and development of efficient business models are of great importance for enterprises. This process requires industrial enterprises to transform their business processes, into modern automation and production units that are compatible with the future by integrating business processes with intelligent systems.

In this sense, computer vision applications are the leading solutions that capture the age and extend to the future. Businesses that have gone through automation and production processes have begun to integrate techniques such as machine learning into their specific operations. The production process is particularly important in this integration since it forms a joint point with the main headings of industrial enterprises such as raw material usage, quality control and operational efficiency. This study proposes an intelligent approach

used to detect faulty products and optimize production efficiency in the industrial area.

The aim of this study is to develop a business intelligence application that detects and separates the defective products that occur during production, helps to determine the cause of the error, contributes to the reduction of the error rate and increase the quality, optimizes the use of raw materials, increases production efficiency and ensures low cost operation of the process by replacing the human factor with computer vision in industrial enterprises producing screws and nuts. Software and frameworks used to design this computer vision application are Python, OpenCV, Tensorflow Object Detection API, Tensorboard, GoogleColab, and Visual Studio Code.

Keywords: Anomaly , Fault, Computer Vision, Deep Learning, Object Detection, Classification.

DERİN ÖĞRENME TEKNİKLERİ İLE ANOMALİ İÇEREN METAL SOMUNLARIN HATA TESPİT VE SINIFLANDIRILMASI

İÇİNDEKİLER

ÜÇÜNCÜ BÖLÜM
EVRİŞİMLİ SİNİR AĞLARI

DÖRDÜNCÜ BÖLÜM
HATA TESPİT VE SINIFLANDIRMA SİSTEMİ

KISALTMALAR

AI	Artificial Intelligence (Yapay Zeka)
API	Application Programming Interface (Uygulama Program Arayüzü)
CNN	Convolutional Neural Network (Evrişimli Sinir Ağı)
ESA	Evrişimli Sinir Ağı
tf	Tensorflow
Mp	Mega piksel
ms	Milisaniye
px	piksel
ROI	Region Of Interest (İlgi Alanı)
RPN	Region Proposal Network (Bölge Öneri Ağı)
s.	Sayfa No
vb.	Ve Benzeri
vd.	Ve Diğerleri
YBS	Yönetim Bilişim Sistemleri (Management Information Systems)
SGYD	Sistem Geliştirme Yaşam Döngüsü
KNN	K En Yakın Komşu
MLP	Multilayer Perceptron (Çok Katmanlı Algılayıcı)
MLR	Çoklu Lineer Regresyon
PR	Polinomsal Regresyon
RF	Rassal Orman
SVC	Destek Vektör Makineleri
GB	Gradient Boosting – Gradyan Artırma
RFC	Random Forest Classifier
AdaBoost	Adaptive Boosting – Adaptif Artırma
DT	Decision Tree – Karar Ağaçları
IoU	Intersection – over – Union
CHT	Circular Hough Tranform – Dairesel Hough Dönüşümü

TABLOLAR LİSTESİ

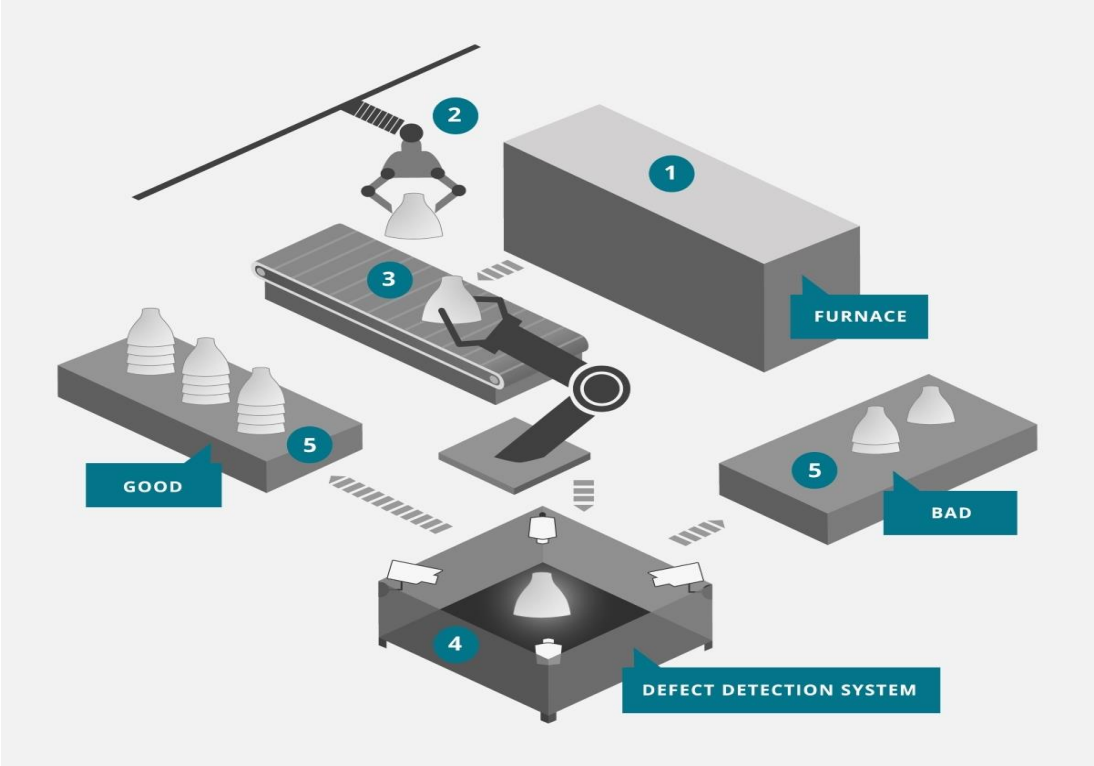
ŞEKİLLER LİSTESİ

EKLER LİSTESİ

GİRİŞ

Günümüzde meydana gelen teknolojik gelişmeler ve dijital dünyada ardarda meydana gelen değişimler serisi, insanlar arasındaki iletişimi ve bilgi paylaşımını kolaylaştırdığı gibi benzer etkiyi endüstriyel alanda da hissettirmiştir. Sanayi devriminden itibaren makineleşen dünya yarı iletken teknolojisinde meydana gelen atılımla birlikte dijitalleşmiş ve otomasyon teknolojisinin gelişmesiyle de insanoğlu, makinelerin iletişim ağındaki mevzilerini yavaş yavaş kaybetmeye başlamıştır. 1990'lı yıllardan itibaren yaygınlaşan internet kavramı bu ağa yeni bir boyut getirmiş ve internet, temas ettiği her nesneyi akıllı hale getirmeyi başarmıştır. Bu aşamadaki başarı ile ortaya çıkan ve Kevin Ashton tarafından 1999 yılında ilk defa dile getirilen "Nesnelerin İnterneti (IOT-Internet Of Things)" ifadesi (Ashton, 2009) ile insanlar arasında var olan etkileşimin bir benzeri de makineler arasında oluşmaya başlamış, bu gelişmelerin kaçınılmaz sonucu olarak da kendi içerisinde özelleşmiş akıllı süreçlerin bileşiminden oluşan Şekil 1'de bir örneği gösterilen akıllı fabrikalar doğmuştur.

Şekil 1: Akıllı Fabrika Süreci



Kaynak: Metaverbis, 2019

İşletmelerde uygulanan akıllı süreçlerin denetiminin yapılması, analizinin yapılması, kalitenin korunması ve verimliliğin artırılması için de yine akıllı sistemlerin günümüzde kalite kontrol sorunları, ürün ve malların imalatında önemli bir rol üstlenmektedir. Gittikçe daha fazla talepkâr olan pazarlarla birlikte şirketler, üretim döngüsü sırasında küçük kusurları bile tespit edebilen daha verimli denetim sistemlerine yatırım yapmak zorunda kalıyorlar. Bilgisayarla görme teknikleri, bu arıza tespit sürecinde önemli bir anahtar sunmaktadır (Gonzalez ve Woods, 1992; Russ,1995; Suetens ve diğerleri, 1992).

BİRİNCİ BÖLÜM
ENDÜSTRİYEL ALANDA BİLGİ SİSTEM GEREKSİNİMLERİ

Hayatın her alanında olduğu gibi, endüstriyel alanda da meydana gelen hızlı paradigma değişimleri, işletmelerin de bu değişime bilgi ve iletişim teknolojilerini yenileyerek eşlik etme gereksinimi duydukları hızlı çözüm bekleyen bir ortamın doğmasına neden olmaktadır. İnsan yığınlarının etkisini kaybettiği makinelerin yükselişini izlediğimiz bu ortamda, kullanıcı dostu arayüze sahip, işlevsel, ekonomik ve akıllı çözümler sunan yenilikçi uygulamalara ihtiyaç duyulmaktadır.

Bu kapsamda incelendiğinde, bilgisayarla görü ile yapay zeka teknolojilerini birlikte oluşturulan çözümlerin endüstriyel işletmelere, akıllı bir şekilde entegre edilmesinin uygun olduğu değerlendirilmektedir. Çünkü üretimde kaynak yönetimi, kalite yönetimi ve elde edilen çıktıların her kademedeki yöneticiye sağlayacağı karar desteği işletmeler için stratejik anlamda büyük değer arz etmektedir.

Kısaca değinilen bu gereksinimleri doğuran problemleri ortaya koymak ve bu aşamadan sonra problemlerin çözümüne yönelik öngörüleri hayata geçirmek yapısal ve stratejik açıdan son derece önemlidir.

1.1. PROBLEMİN TANIMI

Günümüzde, nesnelerin interneti ile birlikte ortaya çıkan ve Endüstri 4.0 ile endüstriyel alana yayılan yeni iletişim yapısı, işletmelerin varlığını sürdürebilmesi için yapısal değişimleri zorunlu kılmış ve her alanda kendini kabul ettiren akıllı değişime uyum sağlamasını da kaçınılmaz kılmıştır.

Akıllı sistemlerin birbirleriyle girdiği etkileşim sonucu doğan sınırsız verinin, ana yakıtı veri olan siber dünya içerisinde kullanılmadan adeta bir siber çöp olarak kaybolmasına izin vermek yerine, verinin anlamlı bir iş parçacığı haline dönüştürülerek kullanılması, endüstriyel işletmelerin kendilerini geleceğe taşımaları için hayati derecede öneme sahiptir.

Bu kapsamda, sektör içerisinde üretime yönelik çalışmalar yapan ve çağa uyum sağlayamayan işletmelerin karşılaştığı:

- Öngörülemeyen ve önlenemeyen üretim hataları,

- Kalite denetiminde yetersizlik
- Üretim sürecinde gecikme,
- Hammadde tüketiminde artış,
- Verimsiz üretim ve kontrol süreciyle birlikte oluşan maliyet artışı,
- Oluşan bilgiden, elde edilmesi muhtemel potansiyel faydanın siber çöplükte yerini alması gibi sorunlar, uygulamanın doğrudan ya da dolaylı olarak katkı sağlayacağı problemler olarak belirlenmiştir.

1.2. LİTERATÜR TARAMASI

Literatür taraması yapılırken, gerçekleştirilen çalışmada uygulanan; bilgisayarla görme, derin öğrenme, transfer öğrenmesi, hata tespiti ve nesne tespiti gibi başlıklara odaklanılarak yapılan çalışmalar incelemeye alınmıştır. Endüstriyel işletmelerin artan ihtiyaçları ve değişen bilgi ve iletişim teknolojilerinin zorunlu kıldığı bu konularda birçok benzer çalışmanın yapıldığı görülmüştür.

Saurabh (2018), önerilen çalışmada makine öğrenmeye dayalı çelik levha kusur algılama sistemi uygulanmıştır. Support vector classifier (SVC-Destek Vektör Makineleri), random forest classifier, gradient boosting, k-nearest neighbours (KNN-k En Yakın Komşuluk), decision trees (karar ağaçları) ve AdaBoost (adaptive boosting) gibi farklı sınıflandırıcılar kullanmıştır. Gradyan Artırma Sınıflandırıcısı (Gradient boosting classifier) uygulanması ve hiperparametre değerlerinin hassas bir şekilde ayarlanmasıyla birlikte ezik, ek, yama, çukurlu yüzey ve çiziklerin %92,5 doğrulukla istenen sınıflandırma değeri elde edilmiştir.

Ferguson ve diğerleri (2018), yapılan çalışma ile metal dökümlerde ve uygulanan kaynak işleminde, meydana gelen kusurların eşzamanlı olarak algılanma ve bölütlenme (segmentasyon) yeteneği ile bir çeşit otomatik kalite kontrol uygulaması önermiştir. Önerilen bu sistem makine öğrenimi için transfer öğrenmesi, veri kümesi artırma (dataset augmentation) ve çok görevli öğrenme (multi-task learning) gibi bir takım güçlü paradigmalar kullanılarak özellik çıkarıcı (feature exactor) olarak ön eğitimli ImageNet ağırlıkları ve ön eğitimli Ms COCO veri seti ile 0.957'lik bir ortalama hassasiyet (mean average precision – mAPbbox) ile geliştirilmiştir.

Campos ve diğerleri (2018), otomotiv sektöründe kullanılan birkaç .çeşit metal bileşen için bir bilgisayarla görme sistemi gerçekleştirmişlerdir. Delikli metal parçalardaki merkezi delikte bulunan tam veya kısmi kapanıklık, civata dişlerinin düzleşmesi ya da kaynak bağlantılarında çapak olması gibi üç tip kusur üzerinde kalite kontrol işlemi gerçekleştirmişlerdir. Farklı gri skala resim analiz algoritmaları birbirinden farklı kusurların her biri için test edilmiştir. Kapalı delik hatası için ilk olarak resimdeki birçok daire şeklini bulan Dairesel Hough Dönüşümü (Circular Hough Transformation) kullanılırken diğer bir yöntem olarak da kontur sınırlama kutusu (bounding box) uygulanmıştır. Bozuk dişler ve çapakların belirlenmesi için ise, iki resim arasındaki piksellerin karşılaştırılarak hatalı bölgenin başarılı bir şekilde belirlenmesi sağlanmıştır.

Hajizadeh ve diğerleri (2016), Hollanda tren rayı ağında, kusurlu ray yüzeylerinin tespit edilmesi için resim verilerinin uygulanabilirliğini test etmeye yönelik bir çalışma yapmışlardır. Bu çalışmayı yaparken verilerdeki dengesizliği engellenebilecek şekilde etiketsiz resimlerin de bulunduğu yarı gözetimli öğrenme yöntemini (semi-supervised learning) uygulamış, hatalı ray resimlerinin bulunduğu bir etiket ile çoğunlukla normal sağlıklı raydan oluşan aynı zamanda kaynaklar, yalıtımlı bağlantılar, iyi huylu kusurlar, küçük çatlaklar gibi potansiyel diğer kusurları da barındıran diğer ray bölümlerini de içeren bir veri seti kullanmıştır.

Xian ve diğerleri (2018), bu çalışmada, metalik bir yüzey için karmaşık endüstriyel senaryolara karşı hem kusur tespit hem de sınıflandırma görevlerini doğru bir şekilde gerçekleştirmek için CNN tabanlı yeni bir mimari sunulmaktadır. Arıza incelemesi, önerilen metoda dayanarak segmentasyon ve sınıflandırma problemine dönüştürülür. Kusurları segmentlere ayırmak ve lokalize etmek için tasarlanan CASAE modülü, kusurlu bir görüntüyü yalnızca hatalı pikselleri ve arka plan piksellerini içeren piksel biçiminde bir tahmin maskesine dönüştürebilir. Kusur kategorisini gerçek muayene ortamlarında hızlı bir şekilde elde etmek için kompakt bir CNN sunulur. Metodumuzun muayene sonucunun IoU puanı, endüstriyel veri seti kullanılarak %89,60 olarak elde edilmiş olup, görsel ve nicel deneysel sonuçlar, önerilen algılama algoritmasının karmaşık endüstriyel ortamın gereksinimlerini karşılamak için yeterli olduğunu göstermiştir.

Staar ve diğerleri (2019), önerilen çalışmada yüzey anomalisi tespiti için ilk defa uygulanan derin metrik öğrenmenin nasıl kullanılabileceğini göstermişlerdir. Burada ilgili probleme, basit bir prototip çıkarım yöntemini derin öğrenme ile güçlendirerek yaklaşılmıştır. Diğer bir deyişle doğrudan piksel şeklinde çıkarım yerine, çıkarım, evrişimli sinir ağı (CNN) tarafından öğrenilen özellik uzayında gerçekleştirilmiştir. Önceki CNN tabanlı yaklaşımların aksine, ağlar, derin metrik öğrenme alanındaki gelişmeleri, yani üçlü ağları kullanarak yüzey dokuları için benzerlik metriklerini açıkça öğrenmek için eğitilmiştir. Kylberg Texture, DAGM ve CIFAR100 olmak üzere üç adet veri seti kullanılmış ve 10 sınıf üzerinden sınıflandırma işlemi gerçekleştirilmiştir. Performansın yüzey tipine bağlı olduğu görülmüştür. 1,3,5 ve 6'ncı sınıfların endüstriyel uygulamalar için kullanışlı olduğu buna karşın önerilen yöntemin 2 ve 4'üncü sınıflar için hatalı ve hatasız alanları ayırt edemediği belirlenmiştir. Ayrıca kullanılan veri setinin de sistem performansında etkisinin yüksek olduğu, bu bağlamda CIFAR-100 veri setinden alınan örneklerin DAGM veri setinden alınan örneklere göre daha performanslı olduğu görülmüştür.

Fernandez-Robles ve diğerleri (2017), yapılan çalışmada rüzgar kulelerinin metal direklerinin işlenmesi için kullanılan freze makinelerinde, direkleri işleyen sitemin milinde bulunan kesici uçları incelemiş, kırık ve kırık değil şeklinde sınıflandırılmasını sağlayan bir sistem önermişlerdir. Uygulamayı gerçekleştirirlerken görüntülerin kontrast kalitesini artırmak ve kenarların algılanmasını kolaylaştırmak için kontrast sınırlı uyarlamalı histogram eşitleme (contrast-limited adaptive histogram equalisation-CLAHE) yöntemini uygulamışlardır. Vidaların dairesel şekillerini tespit etmek için iki aşamalı bir algoritma ile, düz çizgileri tespit etmeyi amaçlayan standart Hough dönüşümünü (Standart Hough Tranform-SHT) temel alarak çevrelerin lokalize edilmesini sağlayan bir özellik çıkarma tekniği olan, dairesel Hough dönüşümünü kullanır (Circular Hough Tranform-CHT). 2 GHz işlemci ile 3 dakikadan daha kısa sürede 1280 x 960px boyutlarında 24 adet görüntüyü analiz edebilen ve sağlam uçların referans görüntüleri ile karşılaştırılmasına gerek duymayan ilk bilgisayarlı görme uygulamasıdır.

Song ve diğerleri (2018), metal vidaların yüzeyindeki mikro seviyedeki kusurları algılamak için derin evrişimli sinir ağı temelinde bir teknik önermiştir. Algılanmaya çalışılan kusurlar arasında yüzey hasarı, yüzey kiri ve ezilmiş vidalar

üzerinde çalışılmıştır. Farklı türdeki kusurlara sahip metal vidaların görüntüleri, tasarlanan derin CNN'leri eğitmek için kullanılan endüstriyel kameralar ile toplanarak CNN modeline uygulanmaktadır. Uygulama sonuçlarına göre, resim başına 1,2 sn'lik ortalama algılama süresi ile önerilen tekniğin %98'lik bir algılama doğruluğu sağladığını göstermiştir. Bu sonuç, örneğin şablon eşleştirme ve LeNet-5 gibi geleneksel makine görme tabanlı tekniklerle karşılaştırıldığında CNN'e dayalı önerilen sistemin üstünlüğünü göstermektedir.

İKİNCİ BÖLÜM
YAPAY ZEKA VE DERİN ÖĞRENME

2.1. YAPAY ZEKA KAVRAMI

Yapay Zeka (Artificial Intelilgence-AI) ilk olarak 1955 (bazı görüşlere göre 1956) yılında Dartmouth kolejindeki iki aylık atölye çalışması çerçevesinde, yeni bir araştırma disiplininin resmi adı olarak kabul edilmiştir. Yapay zeka terimini, 31 Ağustos 1955 tarihinde proje başvurusunda kullanan John McCarthy, yapay zekaya ismini veren kişi olarak kabul edilir (Aydın ve diğerleri, 2019). Yapay Zeka insan gibi düşünmek ve hareket etmek için bir makine sistemi tasarlamaktır. İnsan zekasını yapay olarak taklit etmek ve genişletmek, insan gibi düşünmeyi ve davranmayı amaçlayan makinede gerçekleştirilir (Russell ve Norvig, 2009). Russell ve Norvig (1995) yapay zekayı, sekiz farklı tanımdan yola çıkarak iki ana boyuta göre Şekil 2'de görüldüğü gibi sınıflandırır.

Şekil 2: Yapay Zeka Tanımlamaları ve Sınıflandırılması

Kaynak: Russell ve Norvig, 1995

Birinci boyut, düşünce süreçleri ve mantıkla ilgili, ikinci boyut ise kişiler ve davranışları ele almakla ilgilidir. Derine inildiğinde;

- İnsan gibi davranmak; Turing test yaklaşımıyla,
- İnsan gibi düşünmek; Bilişsel modelleme yaklaşımıyla,
- Akılcı düşünmek; Düşünce kanunlarının yasalarıyla,
- Akılcı davranmak; Akılcı ajan yaklaşımıyla açıklanmaktadır.

Yapay zeka, insanlarda zeka ile ilgili zihinsel fonksiyonları bilgisayar modelleri yardımıyla inceleyip bunları formel hale getirdikten sonra yapay sistemlere uygulamayı amaçlayan bir araştırma alanıdır. "Yapay zeka terimi ilk olarak önemli yapay zeka programlama dillerinden biri olan LISP'i geliştiren ve yapay zeka alanındaki öncülerden biri olan John McCarthy tarafından 1956 yılında ortaya atılmıştır (Russell ve Norvig, 1995: 17-18).

2.1.1. Yapay Zeka Amaçları

Yapay zeka uygulamalarının amacı, insan zekasını örnek alıp, insan zekası olması gereken görevleri gerçekleştirebilecek makineler yapabilmektir. Sonuç olarak insanların bilgisayarlardan daha iyi gerçekleştirdiği görevleri bilgisayarların daha üst düzeyde gerçekleştirmesini sağlamaktır. Genel olarak yapay zeka amaçları üç başlık altında sıralanabilir:

- Makineleri daha akıllı yapabilmek:
    - Geleceğin bilgi toplumunun kurulmasında kilit rol olan "genel bilgi sistemleri" geliştirebilmek.
    - Öğrenme metotlarını formel hale getirmek ve bilgisayarlarda bilgi sistemleri halinde uygulamak
    - Özel bir uzmanlık alanındaki bilgileri, bir bilgi sistemi ya da uzman sistem halinde toplamaktır.
- Zekanın ne olduğunu anlamak:
    - İnsanların zihinsel yetenekleri, bilgi kazanma, öğrenme ve buluş yapmada uyguladıkları strateji, metot ve teknikleri araştırmaktır.
- Makineleri daha faydalı hale getirmek:
    - Yapay zeka iş yardımcıları ve zeki robot timleri geliştirmektir. Birçok davranış biçimi, zekanın belirtileri olarak kabul edilebilir. Aşağıda ise bunun tipik örnekleri belirtilmektedir.

- Tecrübelerden öğrenmek,
- İnsan beyninin fonksiyonlarını bilgisayar modelleri yardımıyla anlamaya çalışmak,
- İnsanların bilgisayar kullanımını kolaylaştırmaya yardımcı insan/bilgisayar ara geçiş birimleri geliştirmek,
- Karışık ve zıt mesajlardan bir anlam çıkartmak,
- Yeni bir duruma başarılı ve hızlı bir şekilde yanıt vermek
- Problemlerin çözümünde muhakeme yeteneğini kullanmak,
- Standart olmayan şaşırtıcı durumlar karşısında, bu durumların üstesinden gelebilmek,
- Bilgiyi anlamak ve kullanmak,
- Düşünmek ve muhakeme etmek.

Yapay zeka programları sürekli gelişmekte ve insan zekası gerektiren durumlara rehberlik etmekte faydalı hale gelmektedir (Reis, 2017:14).

2.2. DERİN ÖĞRENME

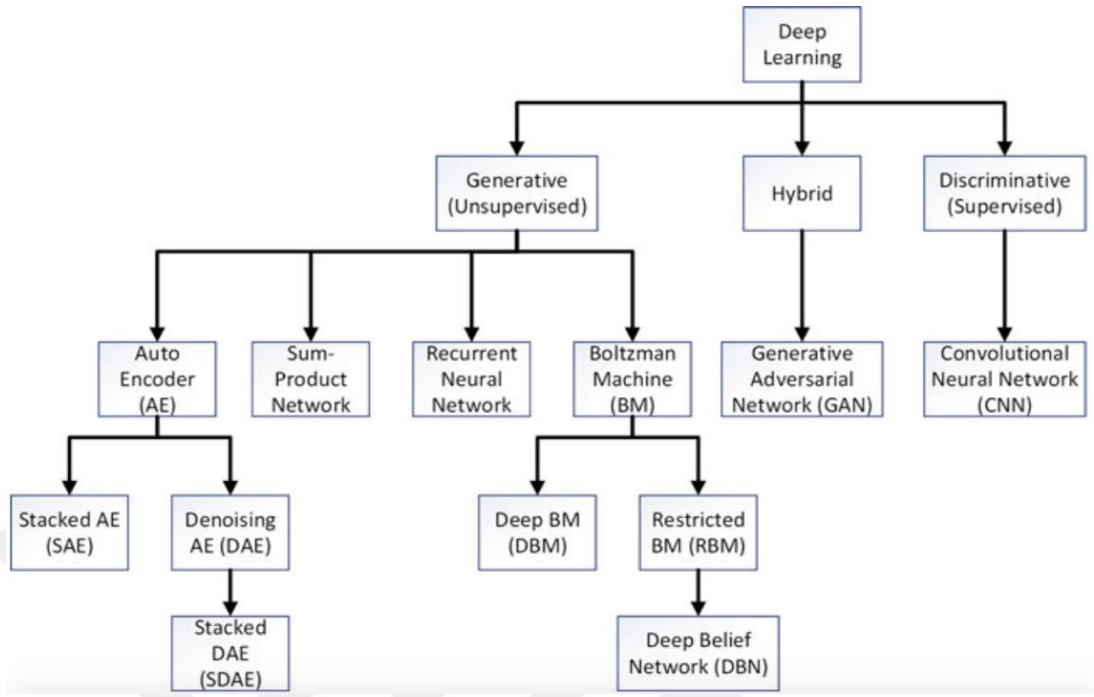
Makine öğrenmesi yapay zekanın bir parçasıdır. Makine öğrenmesi veri ile eğitilebilir. Daha sonra ise verilerden elde ettiği bilgi ile alakalı tahminlerde bulunabilir. Derin öğrenme ise makine öğrenmesinin bir parçasıdır. Büyük veri üzerinde çalışan ön eğitimli (farklı işler için özelleşmiş) çok sayıda yapay zekadan oluşan, çok sayıda düğüme sahip ağlar derin öğrenme olarak isimlendirilir. Derin öğrenme uygulamaları büyük ve etiketsiz veri üzerinde çalışırken öğretmensiz ön eğitim (pre-unsupervised trained), sonrasında öğretmenli öğrenme süreçlerinin çalıştırıldığı sistemlerdir. Bu uygulamaların temel özelliklerinden biri de ağdaki düğüm miktarının çok büyük olmasıdır (Alpaydın, 2011). Derin öğrenme, çoklu işlem seviyelerinden oluşan hesaplama modellerinin, birden fazla soyutlama seviyesine sahip verilerin gösterimini öğrenmesini sağlar. Bu yöntemler nesne tanıma, konuşma tanıma, nesne algılama, görsel nesne tanıma, genomik ve ilaç keşfi gibi diğer birçok alanda son teknolojiyi önemli ölçüde geliştirmiştir. Derin öğrenme, makinenin her katmandaki gösterimini bir önceki katmanda oluşturulan gösterimden hesaplamak için kullanılan dahili parametrelerinin nasıl değiştirilmesi gerektiğinin gösterilmesi için

geri yayılma (back propagation) algoritmasını kullanarak büyük veri kümelerinde karmaşık yapıyı araştırmaktadır (LeCun ve diğerleri, 2015). Derin öğrenme sistemlerinin hem öğrenme hem de karar verme aşamasında çıkarılan özelliklerden yakın olanların bütünleştirilmesi aşaması alt küme (Pooling) oluşturmadır. Milyonlarca farklı verinin (görüntü, ses ve bunun gibi) tüm özellikleri birebir aynı olmayacağı için bu aşama önemlidir. Oluşturulan alt kümelerin stokastik olarak elenmesine silme (drop out) denir. Bu sayede genellikle zayıf ilişkiye ait olan bağlar koparılır. Silme katmanları sadece kıvrımlı ağ için kullanılmazlar, diğer ağlar için kullanılabilir. (Zocca ve diğerleri, 2017). Yapılan işlem gürültü giderme işlemine benzemektedir. Verideki gürültü ise girdi sayısı ile çıktı sayısına eşit olan yapay sinir ağlarıyla (autoencoder) sağlanır. Derin öğrenme sistemlerinde öğrenme ve karar verme süreçleri göz önünde bulundurulduğunda aşağıdaki bileşenlerin her derin öğrenme ağında bulunduğu anlaşılmaktadır.

- Parametreler
- Katmanlar
- Aktivasyon fonksiyonları
- Zarar fonksiyonları
- Optimizasyon yöntemleri
- Hiper parametreler (Patterson ve diğerleri, 2017).

Derin öğrenme başlangıçta sinir ağı algoritmasının ilerlemesinden gelir. Sadece bir sinir ağındaki gizli bir katmanın sınırlamalarının üstesinden gelmek için çeşitli yöntemler uygulanmıştır. Bu yöntemler, hiyerarşik olarak basamaklandırılmış ardışık gizli katmanları kullanır. Derin öğrenmeye ait çeşitli modellerden dolayı, Aminanto ve Kim (2016) derin öğrenmeyi üç alt gruba ayıran, üretici, ayırt edici ve melez olan Deng (2014) tarafından yönlendirilen yaklaşımlara dayanarak birçok derin öğrenme modelini sınıflandırdılar. Sınıflandırma, mimarlık ve tekniklerin (örneğin sentez / üretim veya tanıma / sınıflandırma) amacına dayanmaktadır. Derin öğrenme yöntemlerinin sınıflandırması Şekil 3'te gösterilmektedir.

Şekil 3: Derin Öğrenme Yöntemlerinin Sınıflandırılması



Kaynak: Aminanto ve Kim, 2016

2.2.1. Gözetimsiz Öğrenme

Denetimsiz öğrenme veya sözde üretici model, etiketlenmemiş verileri kullanır. Üretken mimarileri örüntü tanımaya uygulayan temel kavram denetimsiz öğrenme veya ön eğitimdir (Deng, 2014). Daha sonraki ağların daha düşük seviyelerini öğrenmek zor olduğundan, derin üretken yapılara ihtiyaç vardır. Bu nedenle, sınırlı eğitim verilerinden, her bir alt tabakayı, tüm üst tabakalara dayanmadan, tabaka bazında yaklaşımda öğrenmek esastır. Üretken modeller aynı zamanda verilen verilerin ortak istatistiksel dağılımlarını da öğrenmeyi amaçlamaktadır (Deng ve diğerleri, 2014) Bu modeller girdi verilen bağlantı ihtimalini hesaplar ve en yüksek olasılık olan sınıf etiketini seçer (Zang, 2015). Denetimsiz öğrenme olarak sınıflandırılan birçok yöntemler vardır.

2.2.2. Hibrit Öğrenme

Hibrit derin mimari hem üretken hem de ayırt edici mimarileri birleştiriyor. Hibrit yapı, veri ile ayrımcı yaklaşımı ayırt etmeyi amaçlamaktadır. Bununla birlikte, ilk adımda, üretici mimarilerin sonuçları ile önemli ölçüde yardımcı olmuştur. Hibrit mimarinin bir örneği, Derin Sinir Ağıdır (DNN) (Deng, 2014). Deng ve diğerlerine göre (2014), basamaklandırılmış tamamen birbirine bağlı gizli katmanlara sahip çok katmanlı bir ağ olarak tanımlanan DNN, eğitim öncesi bir aşama olarak istiflenmiş bir stokastik tekrarlayan sinir ağı olan Restricted Boltzman Machine (RBM) kullanır. Diğer birçok üretici model, sınıflandırma görevi sınıf etiketleriyle eklendiğinde, ayırt edici veya karma modeller olarak düşünülebilir.

2.2.3. Gözetimli Öğrenme

Denetimli öğrenme veya ayırt edici model, model sınıflandırması için verilerin bazı bölümlerini etiketli verilerle ayırt etmeyi amaçlar (Deng, 2014) Ayrımcı mimarinin bir örneği, özellikle görüntü tanıma için uygun olan özel bir mimariyi kullanan Evrişimli Sinir Ağı olan (CNN)'dir. CNN'nin temel avantajı, el yapımı özellik çıkarımının gerekli olmamasıdır. CNN, çok sayıda veri topluluğundan karmaşık, yüksek boyutlu, doğrusal olmayan eşlemeleri öğrenmek için çok katmanlı ağları eğitebilir (LeCun ve diğerleri, 1998). CNN üç temel kavram kullanır: yerel alıcı alanlar, paylaşılan ağırlıklar ve havuzlama (Nielsen, 2015). CNN kullanılarak başarıyla dağıtılan kapsamlı bir araştırma, Google'dan AlphaGo'dur (Silver ve diğerleri, 2018). Ayrımcı modellerin diğer örnekleri doğrusal ve lojistik regresyonlardır (Wang, 2015).

2.2.4. Derin Öğrenme Kütüphane ve Yazılımları

Derin öğrenme alananda kullanılması amacıyla geliştirilen bir çok kütüphane ve yazılım bulunmaktadır. Tablo 1'de bu kütüphanelerden bazıları hakkında temel bilgilere değinilmiştir.

Tablo 1: Derin Öğrenme Kütüphaneleri

Kütüphane	Yazıldığı Dil	Geliştirici	Öne Çıkan Özellikleri
Theano	Python	MILA Lab	Öğreticileri (tutorial) çok etkili. - Keras, Blocks gibi API sayesinde matematiksel hesaplar kolaylaştırması. GPU desteği
Caffe	Python	Berkeley Vision and Learning Center (BVLC)	Caffe Model Zoo üzerinden indirilebilecek ve hemen kullanılacak önceden eğitimli ağların bulunması. GPU desteği.
Torch	Lua	Ronan Collobert, Clement Farabet,	Algoritmaları oluşturma konusunda maksimum esnekliğe ve hıza sahip olması. GPU desteği. (CUDA) Kullanıcı dostu arayüz
Digits	C++	NVIDIA	Çoklu GPU sistemleri üzerinde sinir ağları tasarımı ve eğitimi, Gelişmiş görselleştirmelerle performansı gerçek zamanlı olarak izleme Tamamen etkileşimli
TensorFlow	Python	Google	Tek bir API ile bir masaüstü, sunucu veya mobil cihazdaki bir veya daha fazla CPU'ya veya GPU'ya dağıtma olanağı.
DeepLearning	Java	Adam Gibson	JVM tabanlı
KNET	Julia	Deniz Yuret	Kolay anlaşılır, kısa kodlama yeteneği. İfade gücü. - GPU Desteği

Kaynak: Cirean ve diğerleri, 2012

Bu kütüphanelerden TensorFlow , Torch, Knet, Caffe ve Theano için bazı veri kümeleri ve modeller üzerinden tek GPU ile çalışma zamanı performansının karşılaştırması Tablo 3'te belirtilmiştir.

Tablo 2: Kütüphanelerin Çalışma Zamanı Performansının Karşılaştırılması

Model	Veri Kümesi	Knet	Theano	Torch	Caffe	TensorFlow
LinReg	**Housing**	2.84	1.88	2.66	2.35	5.92
Softmax	**MNIST**	2.35	1.40	2.88	2.45	5.57
MLP	**MNIST**	3.68	2.31	4.03	3.69	6.94
LeNet	**MNIST**	3.59	3.03	1.69	3.54	8.77
CharLM	**Hiawatha**	2.25	2.42	2.23	1.43	2.86

Kaynak: Cirean ve diğerleri, 2012

Verilen tablo incelendiğinde , en yavaş çalışan kütüphanenin TensorFlow, en hızlı çalışan kütüphanenin ise Theano olduğu görülmektedir (Cirean ve diğerleri, 2012).

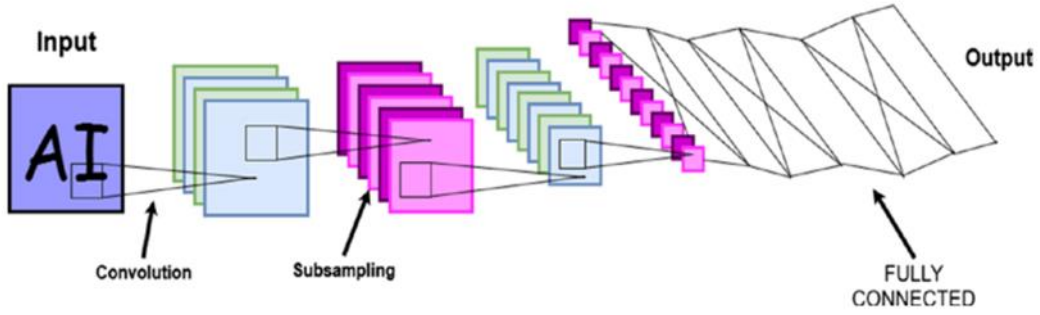
ÜÇÜNCÜ BÖLÜM
EVRİŞİMLİ SİNİR AĞLARI

3.1. EVRİŞİM VE EVRİŞİMLİ SİNİR AĞI KAVRAMI

Evrişim, bir giriş görüntüsünden özellikleri ayıklayan ilk katmandır. Giriş verisinin küçük karelerini kullanarak görüntü özelliklerini öğrenerek pikseller arasındaki ilişkiyi korur. Görüntü matrisi ve filtre veya çekirdek gibi iki giriş alan matematiksel bir işlemdir (Gopikrishna, 2018).

Bir evrişimli sinir ağı (ESA-CNN-Convolutional Neural Network), sinir ağının, iç özellik sunumlarını öğrenerek ve nesne tanıma ve diğer bilgisayarlı görme sorunları gibi ortak görüntü problemlerindeki özellikleri genelleştirerek, hiyerarşik yapıyı koruduğu ileri beslemeli, yapay bir ağdır. Resimlerle sınırlı değildir; aynı zamanda doğal dil işleme problemlerinde ve konuşma tanımada son teknoloji sonuçlar elde etmektedir (Manaswi, 2018). Bir CNN, Şekil 4'te gösterildiği gibi çoklu katmanlardan oluşmaktadır.

Şekil 4: Bir Evrişimli Sinir Ağındaki Katmanlar



Kaynak: Manaswi, 2018

Beyin ağlarını modelleme fikri, bilgisayarların ortaya çıkmasından önce bile bir araştırma sorusu olarak değerlendirilmekteydi. İlk aşamalarda, sinir ağları önerme mantığı ile değerlendirilmekte olup ardından sinir ağlarına uygulanan evrişim ve geri yayılımı gibi kavramların keşfi ile sinir ağları daha iyi sonuç verir hale gelmiştir. GPU'ların ortaya çıkmasına kadar, bilgisayarlar çok katmanlı sinir ağlarını

uygulayacak kadar hızlı değildi. Yani ticari olarak uygun değildi. GPU'ların gücü ve daha verimli algoritmalar sayesinde, CNN'ler gerçek hayattaki uygulamalara hale gelmiştir (Mane ve diğerleri, 2020). CNN'lerin tarihsel gelişimi Tablo 3'te gösterilmiştir.

Tablo 3: CNN'in Gelişim Periyodu

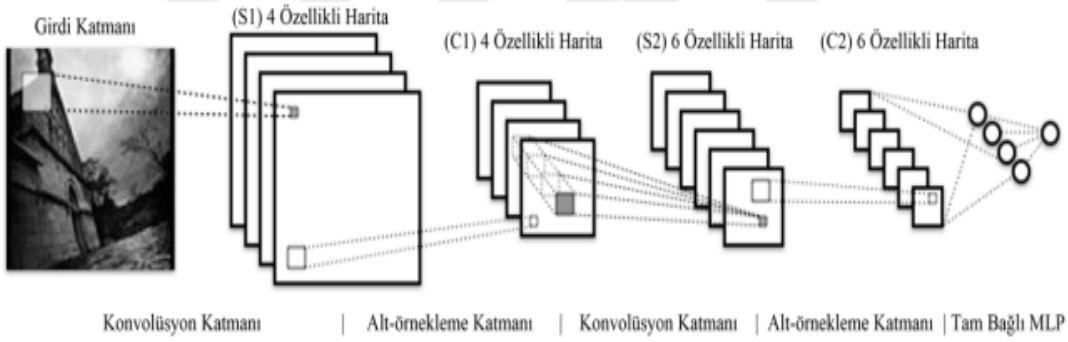
Dönem	Süreç	Yıl	Eylem
1940 - 1979	Sinir Ağının Ortaya Çıkışı	1943	McCulloch, Pits sinirsel aktiviteyi önerme mantığıyla karşılaştırdı.
		1949	Hebb hücre montaj teorisini önerdi.
		1962	Hubel ve Wiesel, kedi beynindeki görsel sistemi modelledi.
1980 – 1998	Evrişim Konsepti	1980	Fukushima, içsel geometrik temsili koruyan kendi kendine öğrenen bir NN önerdi.
		1989	LeCun ve arkadaşları, gerçek hayat uygulamaları için geri yayılımlı (back propagation) CNN kullanımını gösterdi.
1999 - 2010	Daha Etkili CNN'ler	1999	Poggio ve arkadaşları max. Pooling'i önerdi.
		2006	Ranzato ve arkadaşları CNN için maxpooling'i önerdi.
2011 Günümüz	CNN'in GPGPU ivmelenmesi	2011	Ciresan ve arkadaşları GPU'larda CNN kavramını ortaya koydu.
		2012	Hinton ve arkadaşları CNN için Drop Out kullanımını gösterdi.
		2013	LeCun ve arkadaşları daha iyi CNN için Drop Connect'i önerdi.
		2014	Min Lin ve arkadaşları CNN için Ağ içinde Ağ konseptini önerdi.
		2015	Google, CNN için farklı açık kaynak kütüphaneleri yayımladı.

Kaynak: Yazar tarafından derlenmiştir.

Görüntü analizi işlemlerinde en yaygın kullanım alanına sahip derin öğrenme mimarisi olan CNN'ler, bir veya birden fazla evrişim katmanından oluşan ve bu katmanlarda doğrusal olmayan fonksiyonların bulunduğu yapılardır. CNN, bir grup nöronun belirli bir özelliği tanımak ve bunları kategorilere ayırmaktan sorumlu olduğu denetimli bir makine öğrenme algoritmasıdır. Bu algoritmalar, çıktıyı olumsuz etkilemeden girdi özelliklerinin boyutunu azaltmak için girdi üzerinde evrişim gerçekleştirir. Bu özellik görüntüdeki bir kenar veya konuşma alt örneği olabilir. Belirli evrişim çekirdekleri, özellikle görüntülerde, herhangi bir özelliği temsil

etmeyen çok fazla gereksiz pikselin bulunduğu durumlarda, girişin özünü vurgulamak için kullanılır. CNN'in avantajı zengin kusurlu görüntü özelliklerini otomatik olarak çıkarabilmesidir (Chuncheng ve diğerleri., 2019). Bir CNN, elle tasarlanan özelliklere ihtiyaç duymak yerine, eğitim sürecinde otomatik olarak çalışan bir özellik çıkarıcı içerir. CNN özellik çıkarıcısı, eğitim aşamasında ağırlıkları belirlenen özel tür sinir ağlarından oluşur ve aynı boyutta katmanlara sahip standart ileri beslemeli sinir ağları ile karşılaştırıldığında, CNN'lerin çok daha az bağlantı ve parametreye sahip olduğu ve bu sayede eğitilmesinin daha kolay olduğu görülür (Krizhevsky ve diğerleri, 2012). Şekil 5'te verilen LeNet mimarisi, 1988 yılında Yann LeCun tarafından ortaya atılan, ve 1998'lere kadar iyileştirmeleri devam eden ilk CNN ağıdır. LeNet ağında, alt katmanlar art arda yerleştirilmiş konvolüsyon ve maksimum havuzlama katmanlarından oluşur. Sonraki üst katmanlar ise tamamen bağlı geleneksel MLP (Multilayer Perceptron)'ye karşılık gelmektedir.

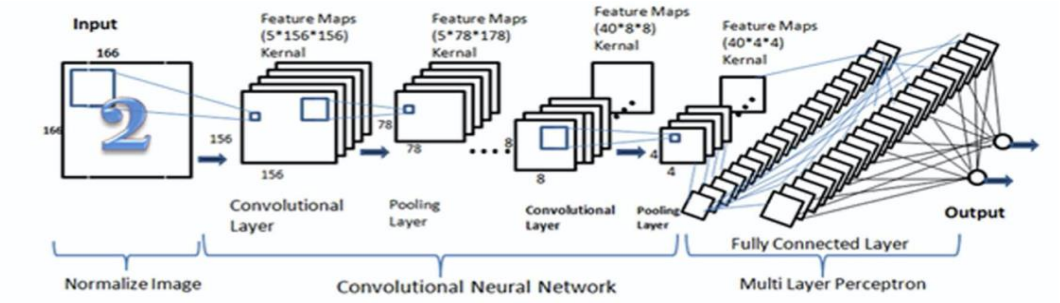
Şekil 5: LeNet Ağının Mimarisi



Kaynak: Cirean ve diğerleri, 2012

Tipik bir CNN mimarisi, Şekil 6'da gösterildiği gibi (LeCun ve diğerleri, 2010), sırasıyla girdi, evrişim, havuzlama, aktivasyon ve sınıflama katmanlarından oluşmaktadır (Chen ve diğerleri, 2018).

Şekil 6: Evrişimli Sinir Ağının Mimarisi

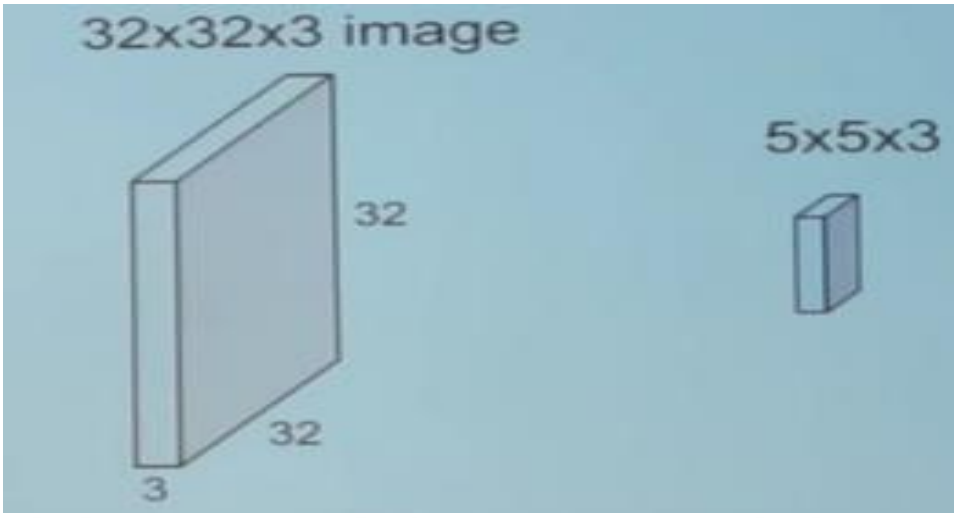


Kaynak: Mane ve diğerleri, 2020

Son yıllarda yapay zeka, bilgisayarla görü ve derin öğrenme alanlarındaki olağanüstü gelişmeler, özellikle de evrişimli sinir ağlarının kullanıldığı, görüntü sınıflandırma ve görü uygulamalarında (Russakovsky, 2015) dikkate değer bir performans artışına yol açmıştır (LeCun, 1989). Bu gelişmeler sonucunda bilgisayarla görü alanı istatistiksel yöntemlerden derin öğrenme yöntemlerine doğru kaymıştır. Derin öğrenme, nesne algılama, hareket izleme, eylem tanıma, insan pozu tahmini ve semantik bölümlendirme gibi çeşitli bilgisayarlı görme problemlerinde büyük adımlar atılmıştır (Volulodimos ve diğerleri, 2018). CNN'ler, yüz tanıma, nesne algılama, otonom araçlar gibi bilgisayarlı görme uygulamalarında son derece başarılı olmuştur.

Ancak önemli bir dezavantajı vardır. Nesne tanıma konusunda evrişimli sinir ağları ile başarılı olunsa da, bazı sorunları da içermektedir. Eğitilmiş bir evrişimli sinir ağı, ilgili nesneye farklı bir açıdan bakıldığında tanıma işlemini farklı başarı oranı ile gerçekleştirmektedir. Evrişimli sinir ağının bir nesneyi oluşturan parçaların arasındaki hiyerarşiyi (örneğin bir yüzün göz, ağız, burun vb. organlardan oluşması) anlayamadığı bilinmektedir. Yapay genel zekaya (artificial general intelligence) giden yolda nesnelerin konum, yönelim ve açısal durumdan bağımsız olarak tanınabilmesi için farklı parametrelerle temsil edilmesi gerekmektedir (Beşer ve diğerleri, 2018). Sinir ağlarından farklı olarak, CNN'lerde girdi Şekil 7'deki gibi çok kanallı bir görüntüye sahip bir vektördür (bu durumda 3 kanallı-RGB).

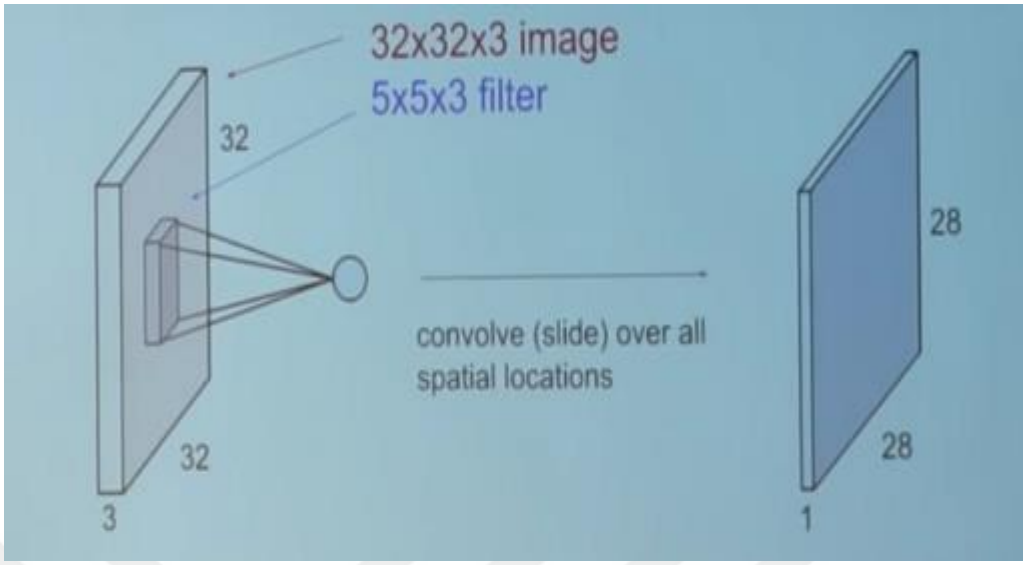
Şekil 7: CNN'e Giren Örnek Bir Resim Bilgisi



Kaynak: Gopikrishna, 2018

Şekil 8'de, 32 * 32 boyutlarında 3 kanallı, yani renkli bir giriş resmi olarak kullanıldığı görülmektedir. Bu görüntünün düzleştirilerek vektör hale getirilmesi gerekmektedir, fakat bu işlem esnasında resimdeki köşeler ve resim derinliği kaybedilmektedir. Buna engel olmak için CNN kullanılmaktadır. CNN ile resmi düzleştirip vektör haline getirmek yerine resmin 3 boyutlu yapısının korunması yoluna gidilir. Ve ağırlık (weight) olarak küçük filtreler kullanılır. Bu örnekte 5x5x3 filtre kullanılmakta ve resmin tüm piksellerine uygulanmaktadır. Filtrenin bize verdiği değer o bölgenin çıkış değeri olacaktır. Şekil 8'de görüldüğü gibi filtre uygulandıktan sonra yalnızca bir çıktı alınmış ve çıktı olarak 28 * 28 * 1 boyutunda bir aktivasyon haritası elde edilmiştir. Layer içinde kaç adet filtre varsa bu filtrelerin hepsi tek tek uygulanır ve sonunda kaç adet filtre uygulandıysa çıkış derinliği de o kadar olacaktır. Ayrıca filtre derinliğinin uygulandığı layer derinliğiyle aynı olmak zorunda olduğuna dikkat edilir.

Şekil 8: Örnek Bir Evrişim İşlemi



Kaynak: Gopikrishna, 2018

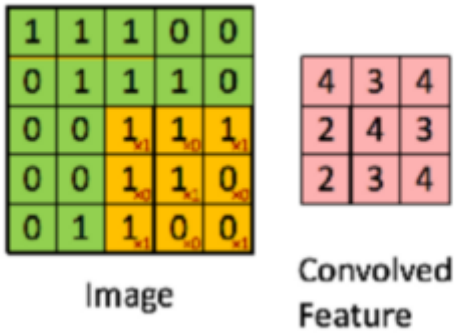
Şekil 9'da görüntü piksel değerleri 0, 1 değerlerinden oluşan 5 x 5 görüntü matrisi ile 3 x 3 filtre matrisinin çarpımı görülmektedir.

Şekil 9: Görüntü ve Filtre Matrisleri

Kaynak: Gopikrishna, 2018

Filtreleme işleminin ardından Şekil 10'da görüldüğü gibi, 3 x 3 boyutlarında bir "Özellik Haritası" oluşur.

Şekil 10: Filtreleme İşlemi Sonucu Oluşan Evrişimli Özellik Haritası



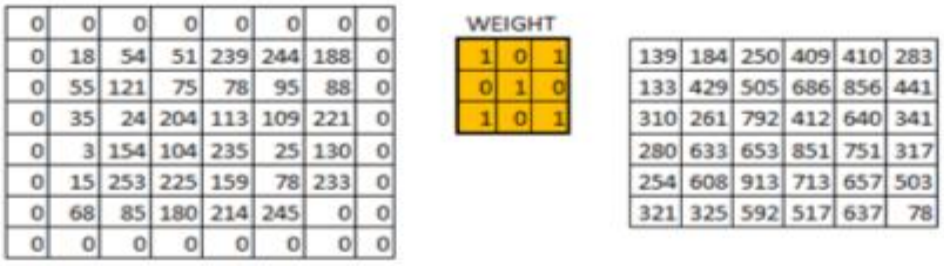
Kaynak: Gopikrishna, 2018

Adım (stride): Adım, giriş matrisine uygulanan filtrenin kaydırıldığı piksel sayısıdır. Adım değeri 1 olduğunda, filtreler tek seferde 1 piksel kaydırılır. Adım değeri 2 olduğunda filtreler tek seferde 2 piksel kaydırılır.

Zero Padding (Sıfır Doldurma): Filtre uygulandığında boyutlar her zaman küçülür. Boyutların her katmanda düşmesini engellemek için zero padding (sıfır doldurma) işlemi uygulanır. Kaybolan her piksele 0 değeri verilirse küçülme engellenmiş olur.

Şekil 11'de, filtreleme uygulanan görüntüden, sıfırlarla doldurma işlemi uygulandıktan sonra görüntünün boyutunun korunduğu gözlemlenebilir.

Şekil 11: Zero Padding Uygulaması



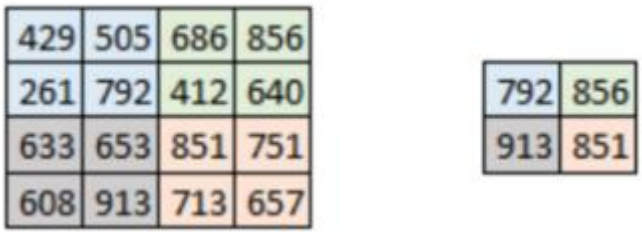
Kaynak: Gopikrishna, 2018

ReLU Aktivasyon Fonksiyonu: ReLU doğrusal olmayan bir operasyon için Doğrultulmuş Doğrusal Ünite (Rectified Lineaar Unit) anlamına gelir. Amacı, doğrusal olmayan gerçek dünya özelliklerini tanıtmaktır ki, bu da hataları kolayca geri

yayabildiğimiz (back propagation) ve ConvNet (Evrişimsel Ağ)'deki ReLU fonksiyonu tarafından etkinleştirilen birden fazla nöron katmanına sahip olabileceğimiz anlamına gelir. Çünkü gerçek dünya verileri, ConvNet'imizin öğrenmesini etkileyen negatif olmayan doğrusal değerler olacaktır.

Havuzlama (Pooling): Havuzlama katmanının amacı, giriş görüntüleri çok büyük olduğunda modeli küçültmek ve parametre sayısını azaltmaktır. Genellikle pooling işlemi ile layer boyutu yarı yarıya indirilir. Daha sonra, sonraki evrişim katmanları arasına düzenli olarak havuzlama katmanlarının eklenmesi arzu edilir. Havuzlama, yalnızca görüntünün uzamsal boyutunu azaltmak amacıyla yapılır. Havuzlama her derinlik boyutunda bağımsız olarak yapılır, yani derinlik boyutunu etkilemez. Bu nedenle görüntünün derinliği değişmeden kalır. Genel olarak uygulanan havuzlama katmanının en yaygın şekli maksimum havuzlamadır (max pooling).

Şekil 12: Max Pooling İşlemi

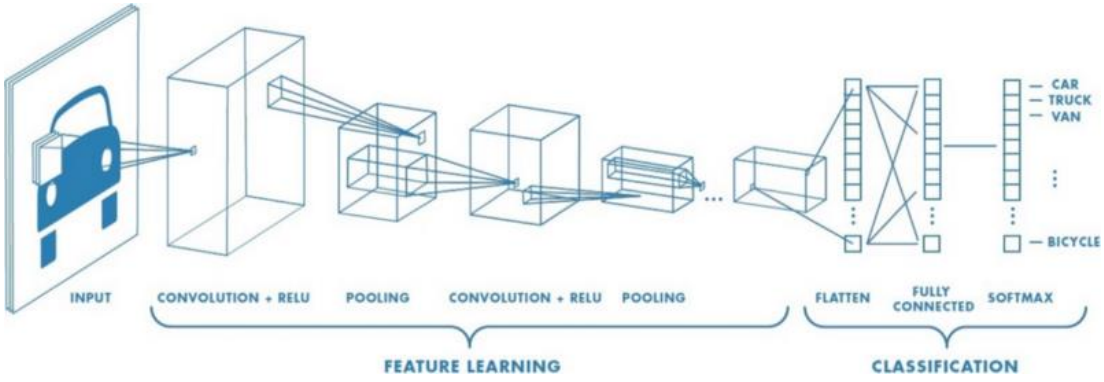


Kaynak: Gopikrishna, 2018

Yukarıda verilen Şekil 12'de, 4 * 4 evrişimli çıkış değeri maksimum havuzlama işleminden sonra 2 * 2 olduğu görülmektedir. 2'şer adım atarken, boyutu da 2 olarak birleştirdik. Maksimum havuzlama işlemi, evrişimli çıktının her derinlik boyutuna uygulanır. Bu katmanda yapılan işlem aşağı örnekleme olarak da adlandırılır. Bu katmanda yapılan işlemler sonucu boyuttaki azalma veri kaybına neden olur. Bunun faydası ise bir sonraki ağ katmanı için hesaplama yükünü azaltmak ve sistemin ezberlemesini engellemektir.

Şekil 13, bir giriş görüntüsünü işlemek için tam bir CNN akışını ve nesnelerin değerlere göre sınıflandırıldığını gösterir.

Şekil 13: CNN Akışı ve Sınıflandırma İşlemi

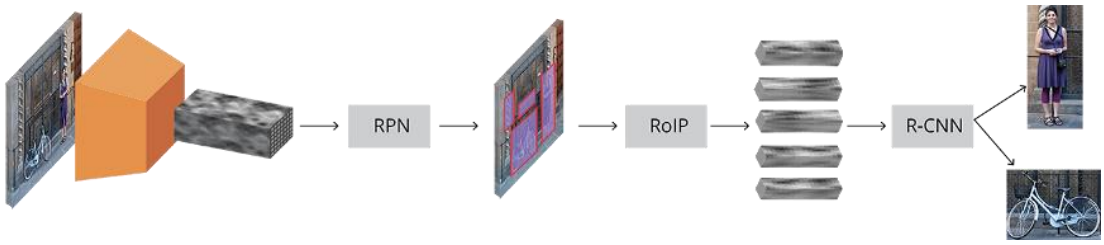


Kaynak: Gopikrishna, 2018

Tamamen Bağlı Katman (Fully Connected Layer) / Çıkış Katmanı: Havuzlama katmanından sonraki özellik eşleme matrisi, vektör (x1, x2, x3…) olarak düzleştirilmiş olacaktır. Bir model oluşturmak için bu özellikleri, fully connected katmanlarla bir araya getirilir. Konvolüsyon katmanları, 3B aktivasyon haritaları oluştururken, görüntünün belirli bir sınıfa ait olup olmadığı gibi çıktıya ihtiyaç duyulur. Çıktı katmanı, tahmindeki hatayı hesaplamak için kategorik çapraz-entropi (cross-entropy) gibi bir kayıp fonksiyonuna (loss function) sahiptir. İleri geçiş tamamlandığında, geri yayılım (back propagation), hata ve kayıp azaltma için ağırlığı (weight) ve önyargıları (biases) güncellemeye başlar. Bu andan itibaren CNN'den bir özellik haritası alındıktan sonra, ilginç bölgeler bulmak için Bölge Öneri Ağı'na (RPN) geçirilmesi gerekir. Bölge Öneri Ağları, bir nesneyi bulup bulmadığına ilişkin kaybı ve nesnenin bulunduğu yere ilişkin kaybı verir.

Şekil 14'te, RPN kullanan ve çalışmamızda uyguladığımız Faster-RCNN için yüksek düzeyli mimari görülmektedir.

Şekil 14: Faster R-CNN Mimarisi



Kaynak: Gopikrishna, 2018

CNN'den giriş görüntüsü için özellikler elde edildikten sonra, Bölge Öneri Ağı (RPN-Region Proposal Network) katmanı ile bölge önerilerinin oluşturulması (Anchors (Çapalar) / Bounding Box (Sınırlayıcı Kutu)) yapılabilir. Öngörülen bölge önerileri daha sonra görüntüyü önerilen bölge içinde sınıflandırmak ve sınırlama kutuları için R-CNN ile offset değerlerini tahmin etmek amacıyla kullanılan bir İlgi Alanı Havuzlama (RoIP-Reigon of Interested Pooling) katmanı kullanılarak yeniden şekillendirilir.

3.2. FASTER R-CNN VE RPN (REGION PROPOSAL NETWORK)

3.2.1. Faster R-CNN ve RPN

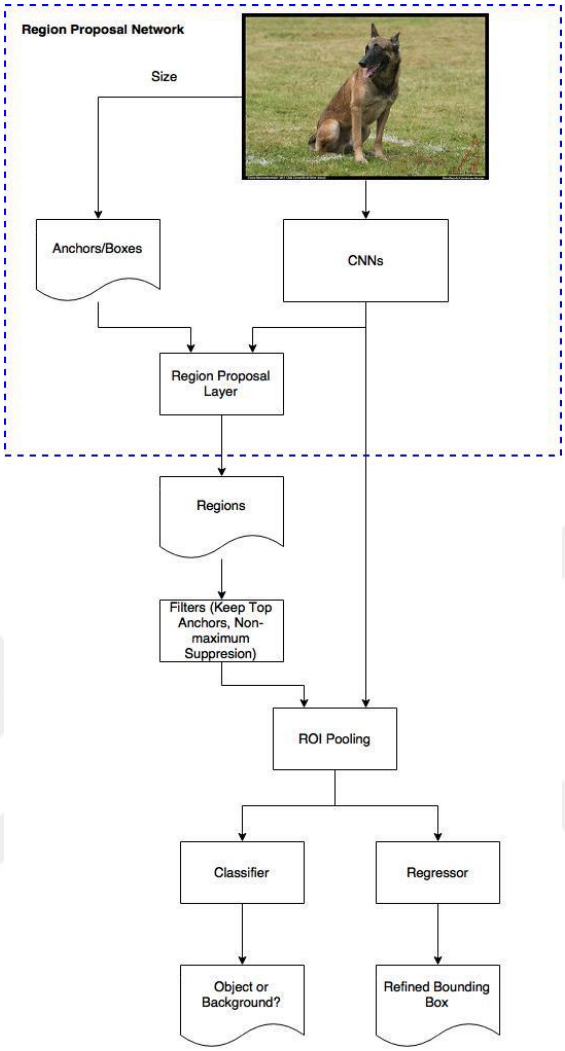
Faster R-CNN Nesne algılama yapan bir ağdır. Adından da anlaşılacağı gibi, temelini oluşturan RCNN ve FastRCNN'den daha hızlıdır. Bu, otonom araçlarda, üretimde, güvenlikte kullanım alanına sahiptir.

Faster R-CNN ağı aşağıdaki temel adımlar çizgisinde çalışır:

- Bir Özellik Haritası (Feature Map) almak için görüntü CNN üzerinden çalıştırılır,
- Etkinleştirme Haritası (Activation Map), Bölge Öneri Ağı (Region Proposal Network - RPN) adı verilen ve ilginç kutular / bölgeler üreten ayrı bir ağ üzerinden çalıştırılır,
- RPN'den gelen ilginç kutular / bölgeler için sınıf + Sınırlayıcı Kutu (class + bounding box) koordinatlarını çıkarmak için birkaç tam bağlantılı katman kullanılır.

Şekil 15'te RPN ve Faster R-CNN Mimarisi gösterilmektedir.

Şekil 15: RPN ve Faster R-CNN Mimarisi



Kaynak: Gopikrishna, 2018

Faster R-CNN'in hızlı olmasının sebebi, bölge önerilerini tahmin etmek amacıyla özellik haritası (future map)'nda seçici arama (selective search) algoritması kullanmak yerine, bölge önerilerini belirlemek için farklı bir ağ kullanmasıdır.

Giriş görüntüsü CNN den geçirilip özellik haritası çıkarıldıktan sonra bu aşamada selective search (seçici arama) ile bölge önerisi almak yerine, bu önerileri ağ içerisinde yapılır. Artık bölge önerileri bu ağ üzerinde yapılır ve hız kazanımı sağlanır. Bundan sonrası Fast R-CNN gibi çalışır.

Sınıflandırma işlemi yapıldıktan sonra 4 farklı parametre ortaya çıkar. Hem bölge önerisi veren ağın hem de normal evrişim işlemlerinin yapıldığı ağın eğitilmesi gerekir.

Burada, RPN'in iki görevi vardır:

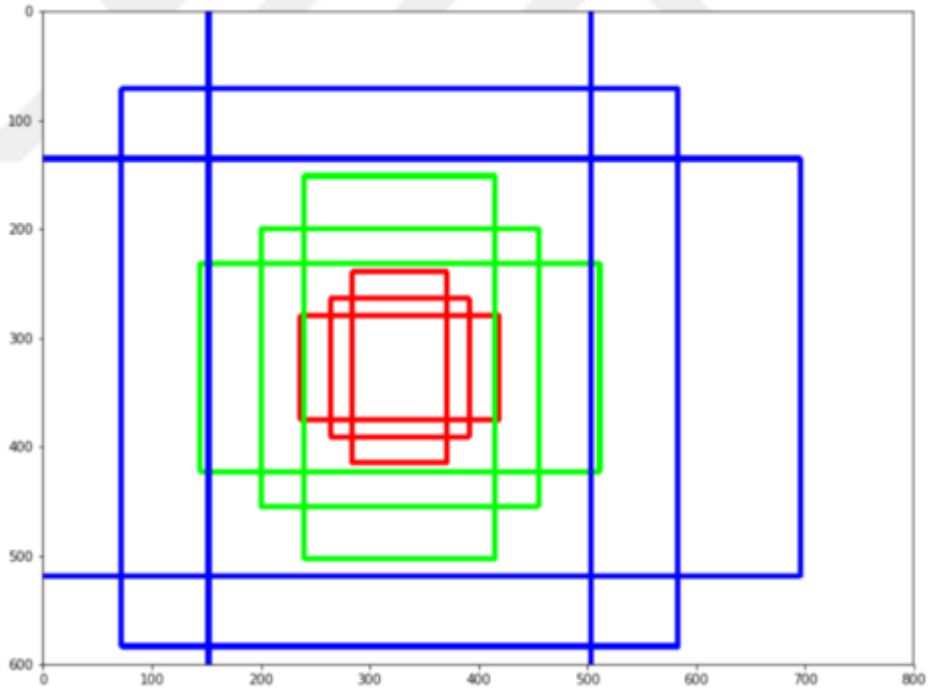
- Her öneri için orada "nesne var mı yok mu?" karar vermek,
- Aynı zamanda önerilerin pencere büyüklüğünü belirlemek.

Asıl Ağımızda yapılacak iki görev vardır:

- Sınıflandırma yaparak baktığı bölge içinde "nesne var mı yok mu?" belirlemek,
- Bulduğu nesnenin sınırlarını belirlemek (Cebeci, 2019).

Son evrişimli katmanda bir dizi evrişimli özellik haritası (özellik matrisi) elde edildikten sonra, bu özellik haritalarında uzamsal olarak kayan bir pencere çalıştırılır. Kayan pencerenin boyutu n × n'dir (burada 3 × 3). Her bir kayan pencere için, hepsi aynı merkeze (xa, ya) sahip, ancak Şekil 16'da gösterildiği gibi 3 farklı en boy oranına ve 3 farklı ölçeğe sahip bir dizi (9) anchor (çapa) üretilir.

Şekil 16: Anchor Üretimi



Kaynak: Gopikrishna, 2018

Daha yakından bakılırsa:

- Üç renk üç ölçeği veya boyutu temsil eder: 128x128, 256x256 ve 512x512.

- Kırmızı kutuları (boxes) / çapaları (anchors) seçelim. Üç kutucuk sırasıyla 1: 1, 1: 2 ve 2: 1 yükseklik genişlik oranlarına sahiptir.

Bu anchor'lar, Pascal VOC veri kümesi ve COCO veri kümesi için iyi çalışır. Ancak, farklı anchor / box tasarımı yapılabilmektedir. Örneğin, yolcuları / yayaları saymak için bir ağ tasarımı yapıldığı varsayılırsa, çok kısa, çok büyük veya kare kutuların düşünülmesi gerekmeyebilir. Düzgün bir anchor seti, hızı ve doğruluğu artırabilir.

Ayrıca, bu anchorların her biri için, bu anchorların yer-doğruluk sınırlama kutuları (GTBox-Ground Truth Box) ile ne kadar örtüştüğünü gösteren bir $\mathbf{p}*$ değeri hesaplanır.

Son olarak, bu evrişim özellik haritalarından çıkarılan 3 × 3 uzamsal özellikler, iki görevi olan daha küçük bir ağı besler: sınıflandırma ve regresyon. Regresörün çıktısı, tahmin edilen bir sınırlama kutusu (x, y, w, h) belirler, sınıflandırma alt ağının çıktısı, tahmin edilen kutunun bir nesne (1) içerip içermediğini veya arka plandan (0 için obje yok) olduğunu gösteren bir p olasılığıdır.

3.2.2. Faster R-CNN ve ROI (İlgi Alanı) Katmanı

RPN'den sonra, farklı boyutlarda önerilen bölgeler elde edilir. Farklı boyutlu bölgeler, farklı boyutlu CNN özellik haritaları anlamına gelir. Farklı boyutlardaki özellikler üzerinde çalışmak için verimli bir yapı oluşturmak kolay değildir. ROI Pooling, özellik haritalarını aynı boyuta indirerek sorunu basitleştirebilir.

Sonuç olarak, farklı boyutlardaki dikdörtgen listesinden, sabit boyuttaki karşılık gelen özellik haritalarının bir listesi hızlı bir şekilde alınabilir.

ROI havuzu oluşturmanın faydalarından biri işlem hızıdır. Çerçevede birden fazla nesne öneri varsa (ve genellikle birçoğu varsa), yine de hepsi için aynı giriş özelliği haritası kullanılabilir. Evrişimleri işlemenin erken aşamalarında hesaplamak çok pahalı olduğundan, bu yaklaşım bize çok zaman kazandırabilir.

3.2.3. Faster R-CNN ve R-CNN Katmanı

Son katman olan bölge tabanlı evrişimli sinir ağı (R-CNN), Faster R-CNN'in boru hattındaki (pipeline) son adımdır. Görüntüden evrişimli bir özellik haritası aldıktan sonra, bunu RPN ile nesne önerileri almak için kullandıktan ve son olarak bu tekliflerin her biri için (RoI Pooling aracılığıyla) özellikleri ayıkladıktan sonra, bu özelliklerin sınıflandırma için kullanılması gerekir. R-CNN, mümkün olan her nesne sınıfı için bir puan çıkarmak için tam olarak bağlı bir katmanın kullanıldığı CNN'lerin sınıflandırılmasının son aşamalarını taklit etmeye çalışır.

R-CNN'nin iki farklı hedefi vardır:

- Önerileri sınıflardan birine ve bir arka plan sınıfına (kötü teklifleri kaldırmak için) sınıflandırır,
- Önerinin sınırlayıcı kutusunu öngörülen sınıfa göre daha iyi ayarlar.

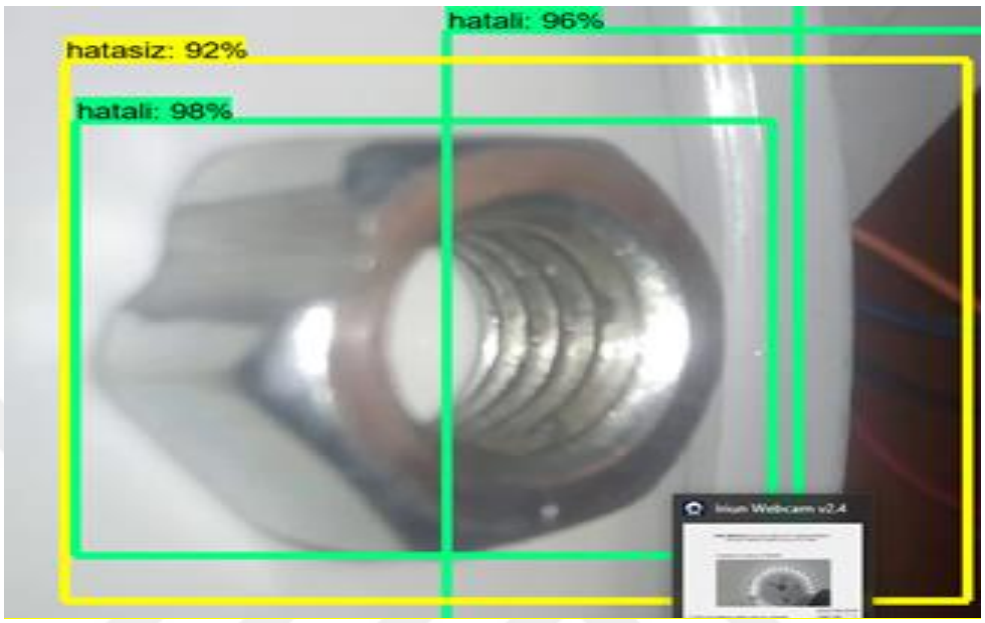
Nesne tespit uygulamalarındaki son gelişmeler, Bölge öneri yöntemleri (Region Proposal Methods-RPN) ve bölgesel tabanlı evrişimli sinir ağları (R-CNN) başarısıyla meydana gelmiştir (Shaoqing ve diğerleri, 2016).

RPN, belirli nesne türlerini tanımlamak yerine resimlerdeki ilgi bölgelerini (ROI'ler) tanımladığı için transfer öğreniminin uygulanması için ideal bir adaydır (Ferguson ve diğerleri, 2018).

RPN, birçok sınıfa sahip büyük bir veri kümesi üzerinde bir nesne algılama ağını eğitirken, RPN, nesne sınıfına göre ayrım yapmadan, muhtemelen nesne içeren görüntünün alt bölümlerini tanımlamayı öğrenir. Bu özellik, ilk olarak nesne algılama sisteminin Micrososft Common Objects in Context (COCO) veri seti gibi çok sayıda nesne sınıfı olan büyük bir veri setinde ön eğtimiyle geliştirilir. İlginç bir şekilde, eğitilmiş nesne algılama sisteminden gelen RPN hatalı bir metal somun görüntüsüne uygulandığında, görüntünün diğer ilginç bölgeleri arasındaki döküm kusurlarını hemen tespit eder.

COCO veri setiyle gerçekleştirilmiş bir eğitimden sonra RPN'in çıktısı Şekil 17'de gösterilmektedir.

Şekil 17: RPN ile İşaretlenen Bir Metal Somun Resmi

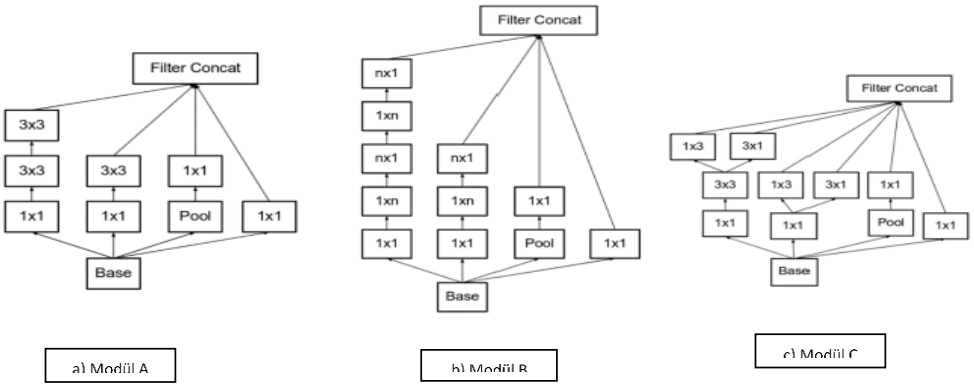


Kaynak: Yazar tarafından derlenmiştir.

3.2.4. Faster R-CNN ve Inception V2 Yapısı

Inception V2, evrişim ağının karmaşıklığını azaltmak için tasarlanmış bir modüldür. Bu modül, evrişim ağının daha derinden geniş olmasını sağlar. Inception V2, A, B, C olarak adlandırılan üç farklı tip modüle sahiptir. Şekil 18 ile gösterilmiştir. Şekil 18 (a) ile gösterilen yumruk modülü (A), 5x5 konvolüsyonun 3x3 konvolüsyon olması yerine değiştirildi. Bu, prensipleri takip ederek, mekansal toplanmanın, temsil gücünde çok fazla veya herhangi bir kayıp olmadan daha düşük boyutlu gömme üzerinde yapılabileceğini gösterdi. 3x3 evrişimi gerçekleştirilerek, evrişim performansı artırılmıştır (C. Szegedy ve diğerleri, 2016).

Şekil 18: Inception V2 Modeli



Kaynak: Alamsyah ve diğerleri, 2019

Filtre boyutunun N x n, 1 x n ve n x 1 evrişimleriyle çarpanlara ayrılması ile, bu yöntemin tekli 3x3 konvolüsyondan %33 daha ucuz olduğu görülmüştür. Bu, modül B olarak Şekil 18 (b) ile gösterilmiştir.

Dahası, filtre genişletilerek daha yüksek boyutlu gösterimler ilkesinin bir ağ içinde yerel olarak işlenmesi daha kolaydır. Genişletilmiş modül, Şekil 18 (c) ile gösterilmiştir (Alamsyah ve diğerleri, 2019).

3.2.5. MSCOCO ve Transfer Öğrenmesi

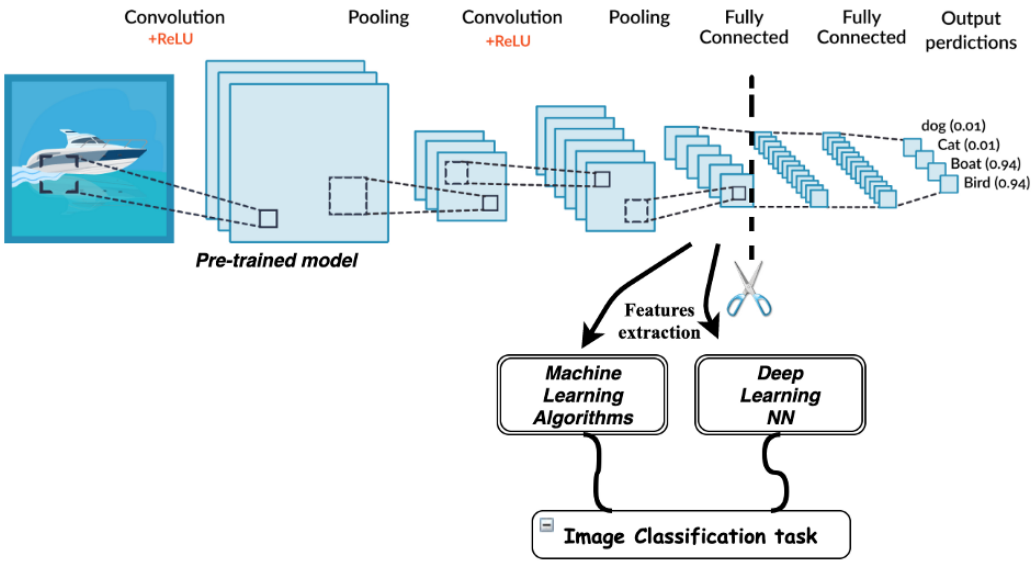
Transfer öğrenmesi, bir ortamda öğrenilen bilgilerin başka bir ortamda genelleştirmeyi geliştirmek için kullanıldığı bir makine öğrenme tekniğidir (Ferguson ve diğerleri, 2018).

Transfer öğrenme, bir sınıflandırma senaryosunda eğitilmiş bir modeli, basit yapısal düzenlemeler yoluyla yeni bir sınıflandırma senaryosuna uyarlar (Gopalakrishnan ve diğerleri, 2017).

(Kolar ve diğerleri, 2018; Gao ve Mosalam, 2018) çalışmaları ile transfer öğreniminin sınırlı eğitim verisi olan alana özgü görevler için özellikle uygulanabilir olduğu gösterilmiştir.

Bu çalışmalardan da anlaşılacağı gibi transfer öğreniminin avantajı, mevcut eğitim veri seti küçük olduğunda bile tespit doğruluğunu artırabilmesidir. Şekil 19, transfer öğrenmesi ilkesini gösterir. Faster RCNN tabanlı yapısal visual inspection metodu çoklu hata tiplerini tespit etmek için önerilmiştir (Cha ve diğerleri, 2018).

Şekil 19: Transfer Öğrenme Şeması



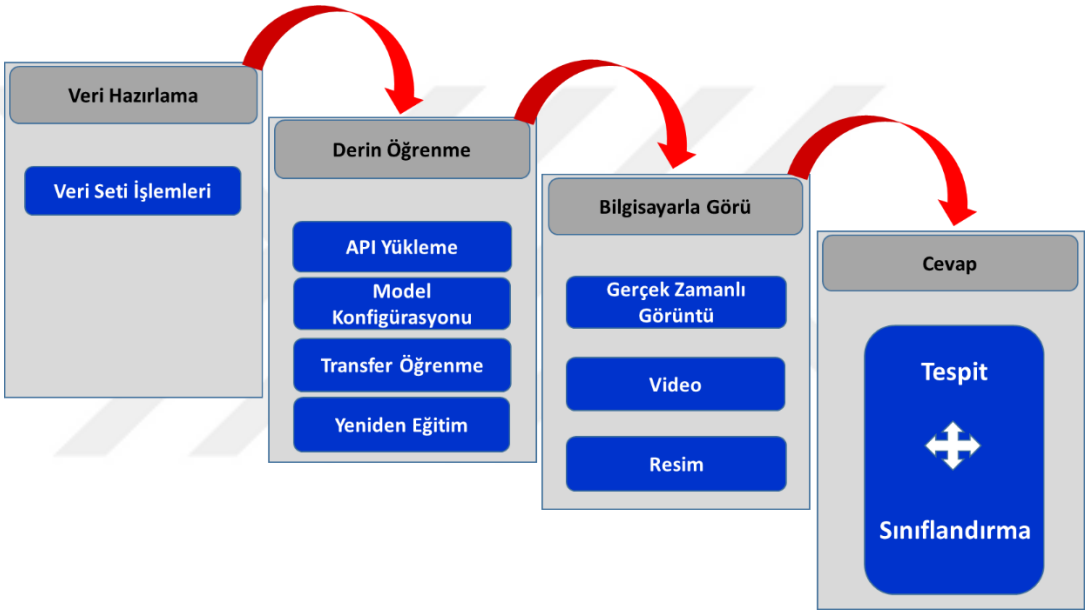
Kaynak: MissingLink, 2016

DÖRDÜNCÜ BÖLÜM
HATA TESPİT VE SINIFLANDIRMA SİSTEMİ

Bu bölümde, derin öğrenme ile resim, video ve gerçek zamanlı kamera görüntüleri üzerinden anomali içeren metal somunlar üzerinde hata tespit ve sınıflandırma işlemi gerçekleştiren bir sistem uygulanmıştır.

Uygulama geliştirilirken Şekil 20'de belirtilen model kullanılmıştır.

Şekil 20: Uygulama Geliştirme Modeli



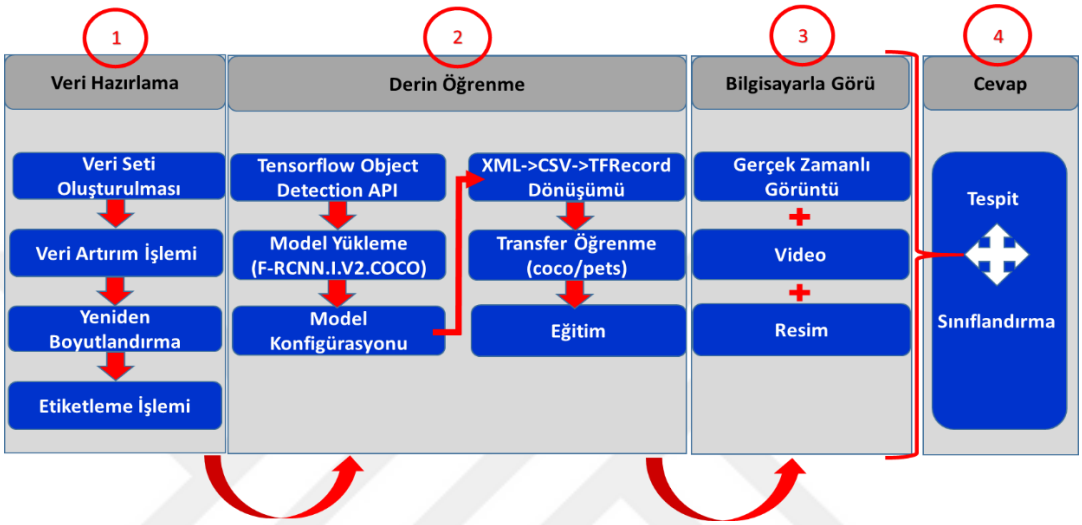
Kaynak: Yazar tarafından derlenmiştir.

Önerilen sistem, metal somunlar üzerinde anomali tespit ve sınıflandırma işlemleriyle bir resim lokalizasyon ve nesne algılama sistemi gibi çalışır. Anomali tespit sisteminin tasarımı, diğer CNN mimarilerine göre doğruluğunun daha yüksek olması nedeniyle Faster R-CNN modeline dayanmakta ve COCO veri seti ile eğitilmiş olan Faster-RCNN Inception V2 modeli kullanılarak transfer öğrenmesi ile sistemin eğitimi gerçekleştirilmiştir.

4.1. METODOLOJİ

Anomali içeren metal somunların hata tespit ve sınıflandırma uygulamasının gerçekleştirilmesi için Şekil 21'de belirtilen iş akışı uygulanmıştır.

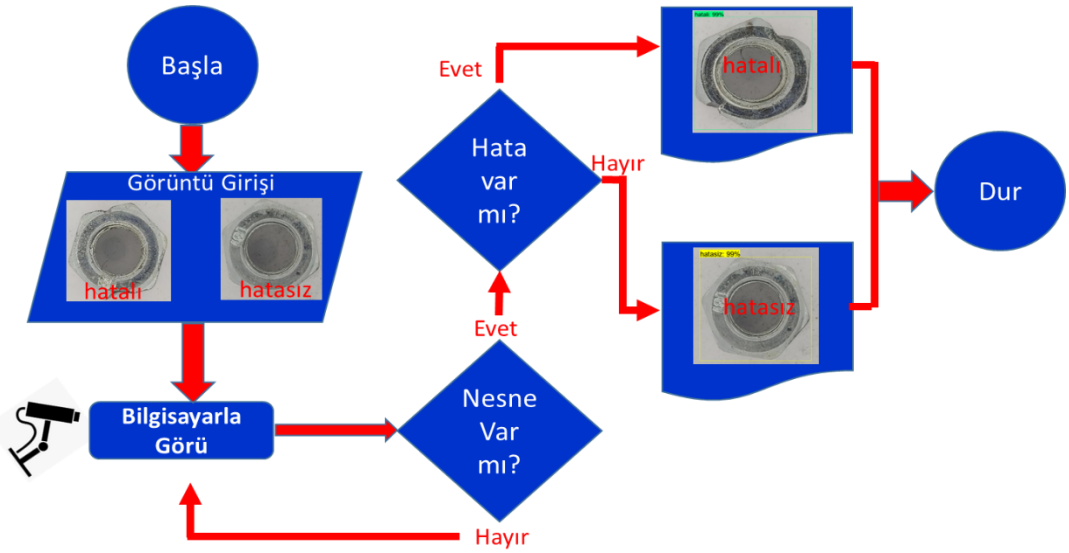
Şekil 21: Hata Tespit ve Sınıflandırma Uygulaması Geliştirme İş Akışı



Kaynak: Yazar tarafından derlenmiştir.

Uygulamanın test aşamasında ve kullanımında ise Şekil 22'de verilen algoritma uygulanmıştır.

Şekil 22: Hata Tespit ve Sınıflandırma Algoritması



Kaynak: Yazar tarafından derlenmiştir.

Veri hazırlama aşamasında, çalışmamıza uygun veri olmadığı için özgün bir veri seti hazırlanmıştır. Veri setindeki veri sayısının artırması ve modelimizin başarısı için veri artırım işlemi gerçekleştirilmiştir. Bu işlemden sonra veri setinin tamamlanmasıyla birlikte metal somunları "hatalı" ve "hatasız" olacak şekilde etiketleme işlemi yapılmıştır.

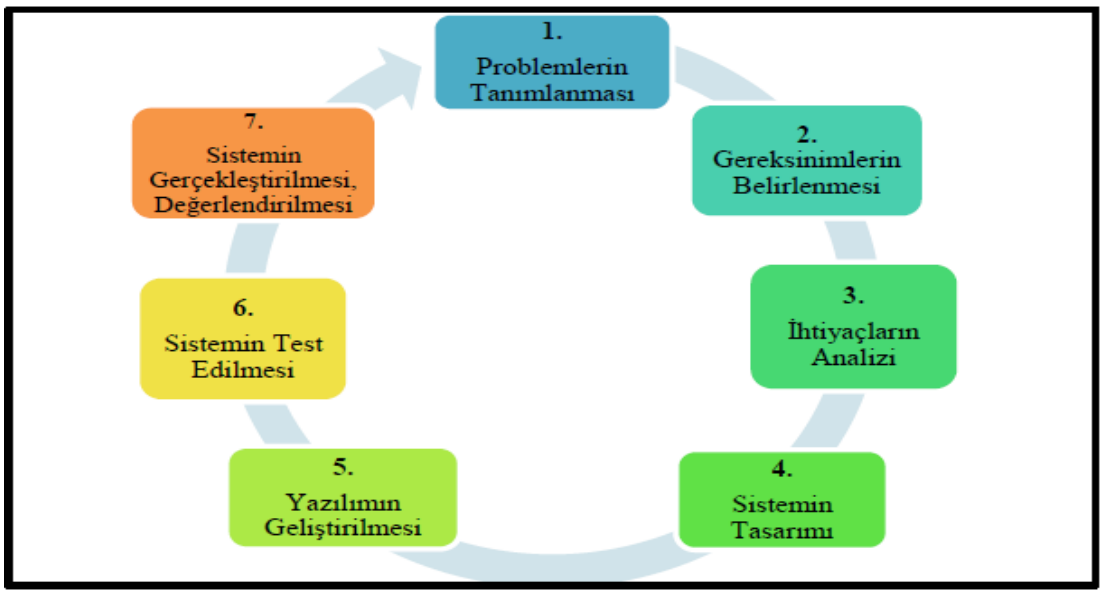
Derin öğrenme aşamasında, Tensorflow Object Detection API kurulumu, bu API'ye ait transfer öğrenmesi gerçekleştireceğimiz, Micrososft Common Objects in Context (COCO) veri setiyle eğitilmiş olan Faster-RCNN-Inception-V2 modelinin yüklemesi ve konfigürasyonu, etiketleme işleminin ardından oluşan etiket bilgilerinin tensorflow'u anlayacağı tfrecord dosyalarına dönüştürülme işlemi ve ardından modelin özgün veri setiyle eğitimi gerçekleştirilmiştir.

Bilgisayarla görü aşamasında, nesne tespit yazılımlarımız çalıştırılarak resim ve video dosyaları üzerinden ya da gerçek zamanlı kamera görüntüleri üzerinden görüntü alma işlemi gerçekleştirilmiştir.

Cevap aşamasında ise uygulamamız gördüğü metal somunları tespit ederek "hatalı" veya "hatasız" olarak sınıflandırma işlemini başarmıştır.

Hata tespit ve sınıflandırma uygulaması geliştirilirken, Şekil 23'te belirtilen ve bir disiplin içeren Sistem Geliştirme Yaşam Döngüsü (SGYD) adımlarına bağlı kalmaya özen gösterilmiştir.

Şekil 23: Sistem Geliştirme Yaşam Döngüsü



Kaynak: Tecim, 2015: 3.

Problemin ve amaçların tanımlanması aşamasında, hedef sektörün iş akışları incelenmiştir.

Belirlenen eksiklere yönelik gereksinimler ortaya konmuştur.

İhtiyaçların analizi kapsamında ise ihtiyaç duyulduğu düşünülen geliştirme süreçleri ve yöntemler ile ihtiyaç duyulan veriler belirlenmiştir.

Sistem tasarımında, hata tespit ve sınıflandırma işleminin sorunsuz yapılabilmesi için, metal somunların konveyörde ilerleyiş biçimleri, kullanılacak kamera sayısı ve yakalanacak görüntü açısına uygun bir veri seti ile uygun bir uygulama tasarımına gidilmiştir.

Yazılım geliştirme aşamasında ise derin öğrenme uygulamaları için en uygun dillerden Python kullanılmıştır. Ayrıca, ihtiyaç duyulan kütüphaneler ve frameworkler kullanılarak özellikle, ücretsiz GPU desteği sağlaması sebebi ile, Google'ın bulut hizmeti olan Colaboratory kullanılmıştır.

Sistem geliştirilirken, eğitim aşamasına kadar uygulanan yapılandırmaların doğru olup olmadığı ve eğitim ardından uygulamanın sınıflandırma işlemini doğru yapıp yapmadığı test edilmiştir. Alınan hatalar sonucunda, konfigürasyon işlemleri tekrar düzenlenerek yeniden eğitim sağlanmış ve başarı oranı artırılmıştır.

Sistemin gerçekleştirilmesi ve değerlendirilmesi aşamasında, sistemin işletmenin kullanımına hazır hale getirilmesi hedeflenmiştir.

4.2. YAPILAN HAZIRLIKLAR

Çalışmaya başlanmadan önce içerisinde metal somunların yer aldığı bir veri seti olmadığı için model üzerinde istenilen eğitimi yapabilmek amacıyla özgün bir veri seti hazırlama gereği duyulmuştur.

Hatalı ve hatasız somunların yer aldığı bir veri seti oluştururken uygun nitelikte görüntüler elde etmek için uygun ortamın hazırlanması yapılması gereken ilk iş olarak ele alınmıştır.

4.2.1. İdeal Ortamın Hazırlanması

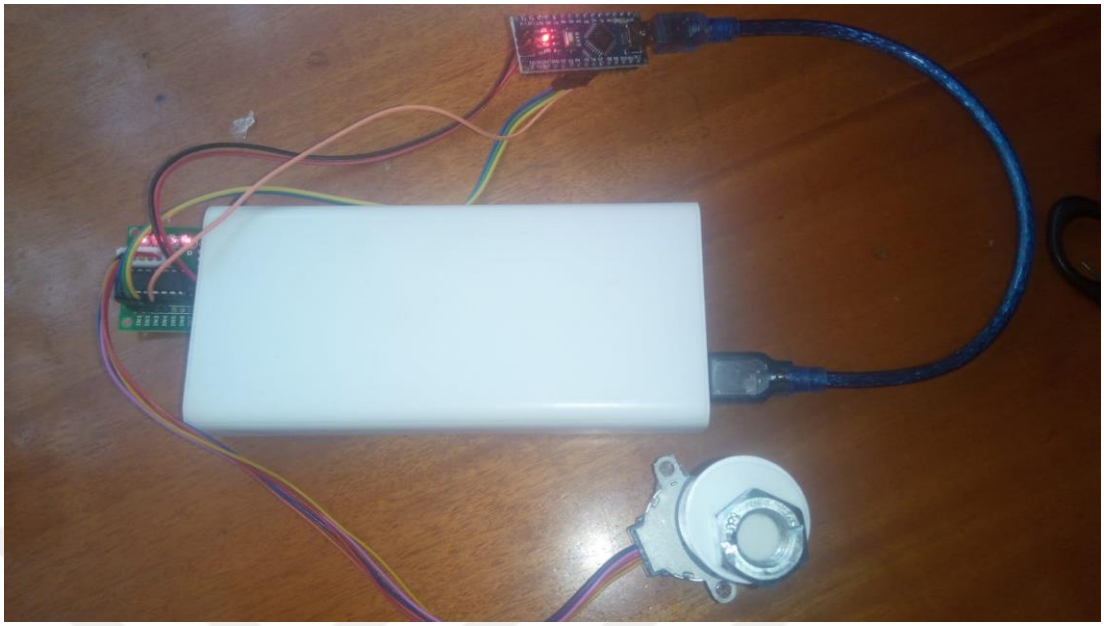
Veri setini hazırlarken alınan görüntülerin ihtiyaç duyulan ideal ortam, bir plastik kabın şerit ledlerle aydınlatılması ve görüntü alınması için yan ve üst taraflarından delik açılmasıyla birlikte Şekil 24'te görüldüğü gibi adeta küçük bir stüdyo şeklinde oluşturulmuştur.

Şekil 24 : Oluşturulan Stüdyo Ortamı

Kaynak: Yazar tarafından derlenmiştir.

Ardından ihtiyaç duyulan açılardan el değmeden ve zaman kaybetmeden ardarda seri görüntüler alınabilmesi için oluşturulan stüdyoya step motor devresi eklenmiştir. Step motor, step motor sürücü kartı ve Arduino nano kartı kullanılarak oluşturulan Şekil 25'te görülen sistem ile step motorun 200 ms gecikmelerle adım adım dönmesi sağlanmış olup, step motor üzerine eklenen bir stand ile de somunların hareketli olarak her açıdan görüntülerinin alınması sağlanmıştır. Step motor bilgileri ve kullanılan kod parçası Ek 1'de belirtilmiştir.

Şekil 25: Step Motor Çalışma Sistemi



Kaynak: Yazar tarafından derlenmiştir.

İdeal ortamın sağlanmasının ardından, öncelikle 13 Mp kamera çözünürlüğe sahip Lenovo K5 Note akıllı telefon ile 1080 px video çözünürlükte videolar çekilmiş ve VLC medya oynatıcı uygulaması ile bu videolardan, belirtilen frame aralıklarında farklı açılardan hatalı ve hatasız görüntülerin yer aldığı 360 adet resim hazırlanmıştır. Bu veri setinde yer alan 360 adet resim 72 adet test ve 288 adet train olacak şekilde ayrıştırılmıştır. Elde edilen bu resimler, derin öğrenme modeline verildiğinde harcanan eğitim süresinin kısaltılması için Photoshop CS2 Extended grafik tasarım uygulaması kullanılarak en/boy oranları 100x76 px çözünürlüğüne indirgenmiştir. Daha sonra, kullanılan derin öğrenme modellerinin veri giriş değerlerine uyum sağlaması ve herhangi bir veri kaybına mani olmak amacıyla bu değerler Python ile yazılmış olan ve Ek-2'de belirtilen kodlar kullanılarak 600x600 px çözünürlükte standart boyutlara getirilmiştir.

Ardından veri setinin yetersiz olduğu düşünülerek 12 Mp kamera çözünürlüğüne sahip **LG G7 ThinQ** akıllı telefon kullanılarak, 1:1 görüntü oranında, 3492x3492 px çözünürlüğünde resimler çekilerek, aralarında nesne algılama uygulamamıza uygun olduğu değerlendirilen yalnızca üst ve yan açılardan alınmış olan resimler seçilmiş, somun resimlerinin sayısını artırmak amacıyla hatasız somun resimlerine veri artırım (data augmentation) işlemi uygulanarak toplam 2000 adet

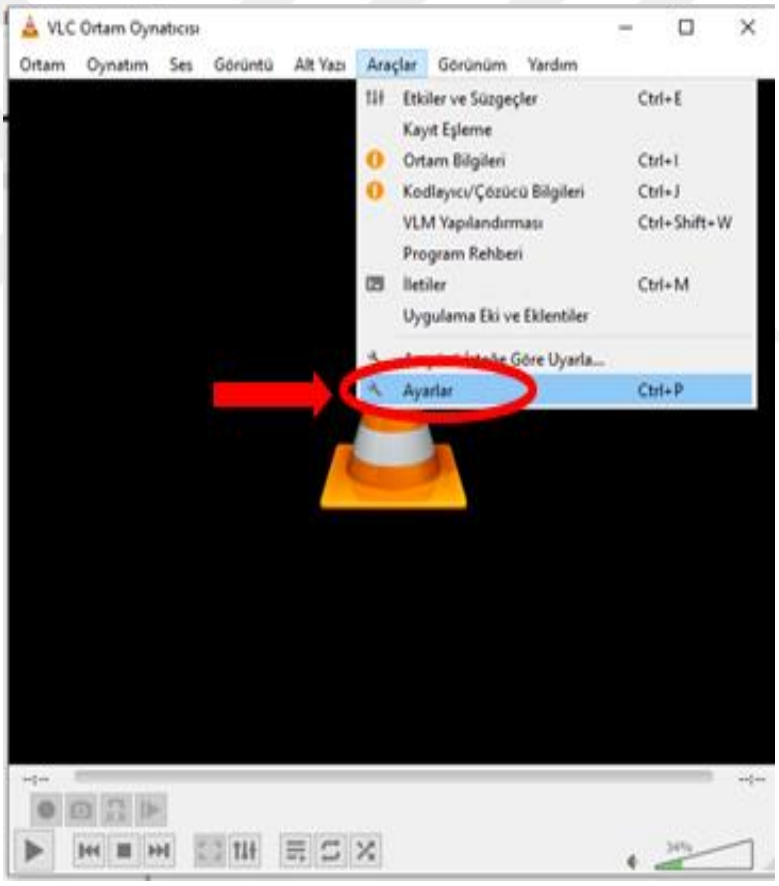
resimden oluşan özgün veri setimiz hazırlanmıştır. Veri setimizdeki 2000 adet resim 400 adet test ve 1600 adet train olmak üzere iki gruba ayrılmıştır.

4.2.2. VLC Media Player Uygulamasının Yapılandırılması

Video üzerinden istenilen frame aralıklarında resim alınabilmesi için VLC Media Player'ın, aşağıdaki adımlarda belirtildiği gibi yapılandırılması sağlanmıştır:

- VLC ortam oynatıcısının arayüzü açılır ve Araçlar menüsünden Ayarlar sekmesi seçilir.

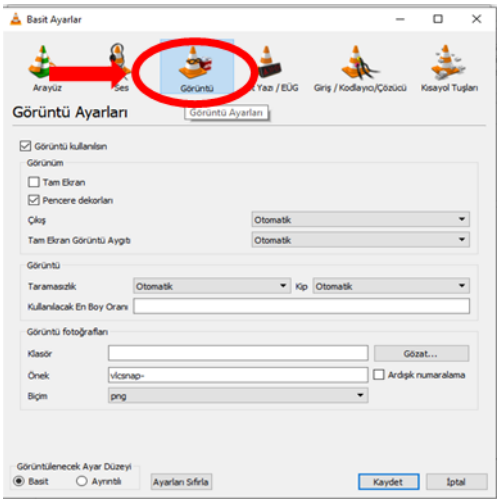
Şekil 26: VLC Player Konfigürasyon Ayarları



Kaynak: Yazar tarafından derlenmiştir.

- Video ve resim işlemleriyle ilgilendiğimiz için Açılan Basit Ayarlar penceresinden "Görüntü" menüsü seçilir.

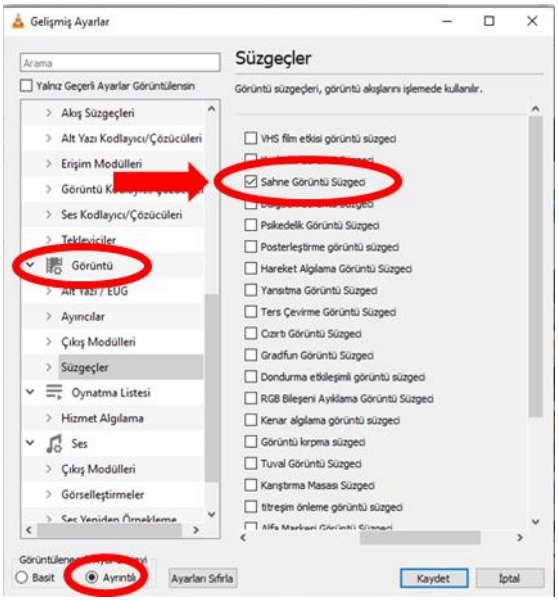
Şekil 27: VLC Player Görüntü Ayarları



Kaynak: Yazar tarafından derlenmiştir.

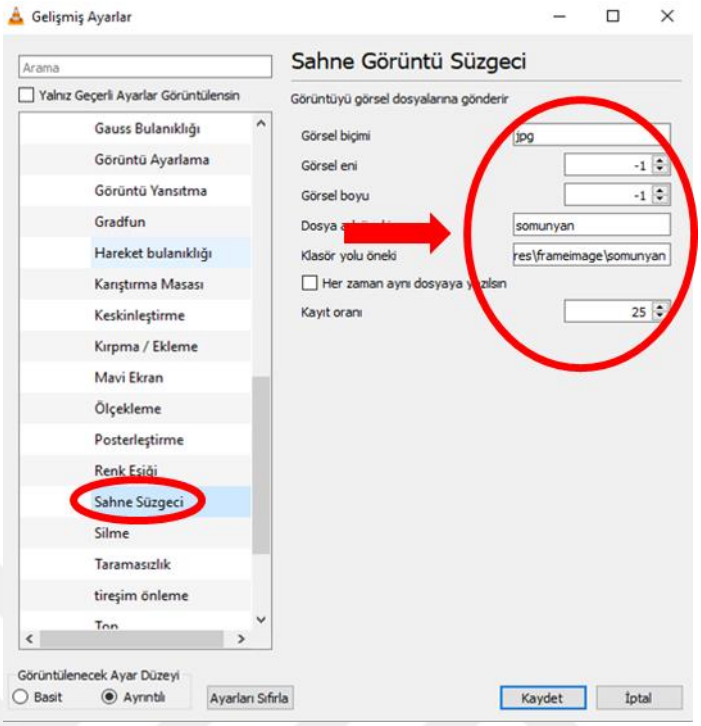
- Görüntü alma işlemlerine giriş yapabilmek için pencerenin sol alt tarafında yer alan "Ayrıntılı" seçeneği seçilerek açılan "Gelişmiş Ayarlar" penceresinden Görüntü" menüsü ve bu menü içerisinden de "Sahne Görüntü Süzgeci" işaretlenir.

Şekil 28: VLC Player Sahne Görüntü Süzgeci



Kaynak: Yazar tarafından derlenmiştir.

Şekil 29: VLC Player Sahne Görüntü Süzgeci Konfigürasyonu



Kaynak: Yazar tarafından derlenmiştir.

- Sahne Süzgeci Ayarlarından "Sahne Görüntü Süzgeci" değerleri girilmelidir.

Buradan kayıt edilecek resmin formatı, boyutları, görüntü alınması istenen videonun bulunduğu klasör uzantısı, kaydedilecek olan resmin kaydedilmesi istenen klasör yolu ve Videonun frame değerine göre istenen kayıt periyodu girilir. Kullandığımız videonun frame değeri 25 olduğu için biz burada 25 değerini kullandık ve her saniyede 1 kare resim kaydedilmesini sağladık.

4.2.3. Veri Artırma (Data Augmentation) İşlemi

Sinir ağlarının derin öğrenme performansı, mevcut veri miktarının sık sık artırılmasıyla iyileşme kaydeder. Veri artırımı, mevcut eğitim verilerinden yapay olarak yeni bir eğitim verisi oluşturmak için kullanılan bir tekniktir. Bu, yeni ve farklı eğitim örnekleri oluşturan eğitim verilerinden örneklere alana özgü teknikler uygulanarak yapılır.

Resim veri artırma işlemi, belki de en iyi bilinen veri artırma türüdür ve orijinal görüntüyle aynı sınıfa ait eğitim veri seti içindeki görüntülerin dönüştürülmüş versiyonlarını oluşturmayı içerir.

Bu dönüşümler, kaydırma, çevirme, yakınlaştırma ve çok daha fazlası gibi görüntü işleme alanındaki birçok işlemi içerisinde barındırır.

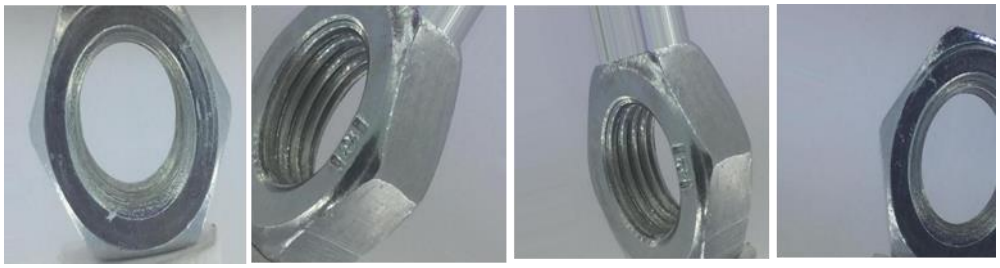
Amaç, eğitim veri setini yeni ve makul örneklerle genişletmektir. Bu, eğitim seti görüntülerinin model tarafından görülmesi muhtemel olan varyasyonları anlamına gelir. Örneğin, bir kedi resminin yatay bir şekilde çevrilmesi, fotoğrafın soldan veya sağdan çekildiği için mantıklı gelebilir. Bir kedi fotoğrafının dikey olarak çevrilmesi anlamsızdır ve muhtemelen modelin baş aşağı bir kedinin fotoğrafını görmesi pek mümkün olmadığından uygun olmayabilir.

Bu nedenle, bir eğitim veri seti için kullanılan özel veri artırma tekniklerinin seçiminin dikkatli bir şekilde ve eğitim veri seti bağlamında ve problemin alanı bilgisi dahilinde seçilmesi gerektiği açıktır.

Evrişimli sinir ağı veya CNN gibi modern derin öğrenme algoritmaları, görüntüdeki konumlarına göre değişmeyen özellikleri öğrenebilir. Bununla birlikte, veri artırma bu dönüşüme daha fazla yardımcı olabilir ve öğrenmeye karşı değişmez yaklaşımı destekleyebilir ve sıralama, fotoğraflardaki ışık seviyeleri ve daha fazlası gibi dönüşümler için değişmeyen öğrenme özelliklerinde modele yardımcı olabilir.

Görüntü veri büyütme işlemi genellikle eğitim veri setine uygulanır, doğrulama veya test veri setine uygulanmaz. Bu, görüntü yeniden boyutlandırma ve piksel ölçeklendirme gibi veri hazırlığından farklıdır; modelle etkileşime giren tüm veri kümelerinde tutarlı bir şekilde gerçekleştirilmeleri gerekir (Brownlee, 2019).

Şekil 30 : Veri Artırımı Sonucu Oluşan Resim Örnekleri



Kaynak: Yazar tarafından derlenmiştir.

Veri artırma işlemi, keras kütüphanesinin ImageDataGenerator() sınıfına ait bazı metotlar kullanılarak gerçekleştirilmiştir.

Amacımız yetersiz olduğu düşünülen hatasız somun resimlerinin sayısının ve çeşitliliğinin artırılarak eğitim doğruluğunun artırılmasını sağlamaktır.

Data augmentation için Ek-3'te belitilen kodlar kullanılmıştır.

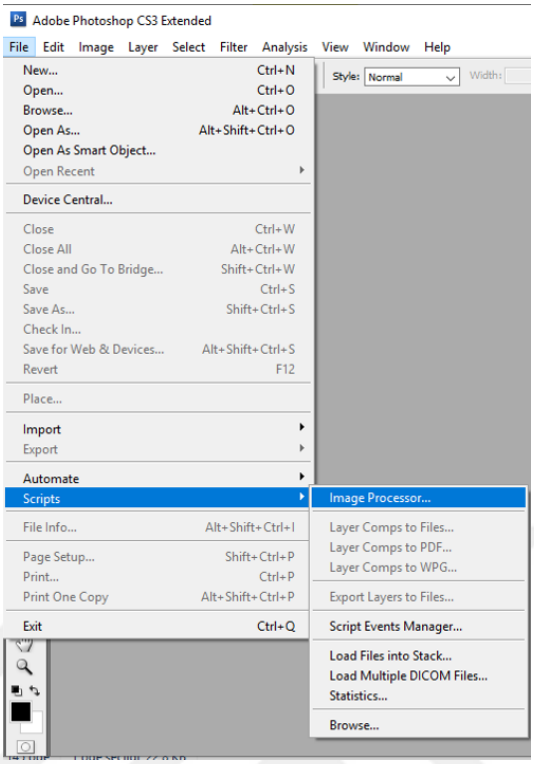
4.2.4. Resim Boyutlarının Düzenlenmesi

4.2.4.1. Photoshop ile Resim Boyutlarının Düzenlenmesi

Eğitimin daha hızlı gerçekleştirilebilmesi için oluşturulan ilk veri setinde kullanılması düşünülen 360 adet resmin boyutlarının düşürülmesi için Photoshop CS3 Extended uygulaması aşağıdaki adımlar kullanılarak yapılandırılmıştır :

- Photoshop programından File -> Scripts -> İmage Processor adımları izlenerek Image Processor penceresi açılır.

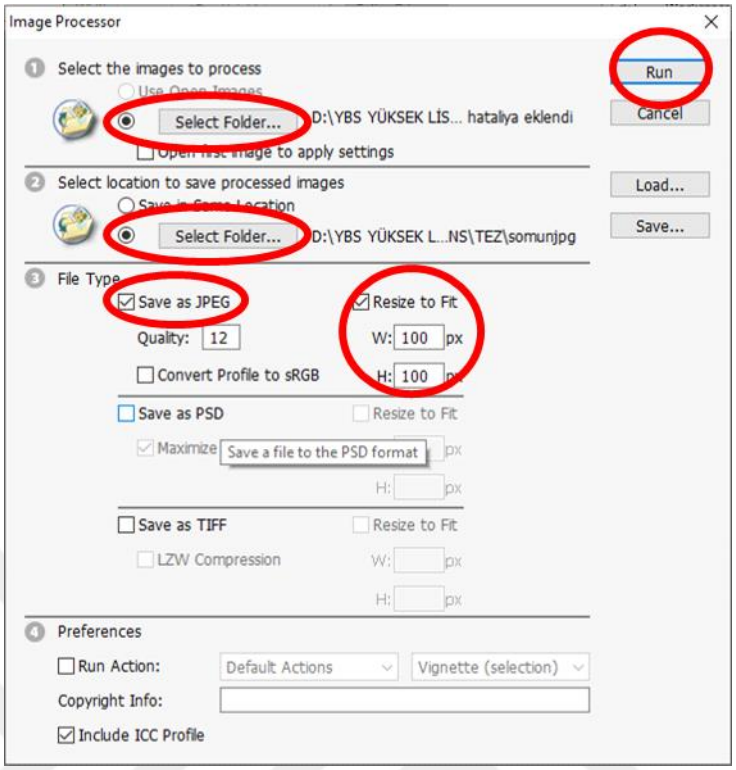
Şekil 31: Image Processor Penceresinin Açılması



Kaynak: Yazar tarafından derlenmiştir.

- Açılan Image Processor penceresinde sırasıyla :
  - İşlem yapılacak resimlerin bulunduğu klasör seçilir.
  - İşlem yapılan resimlerin kaydedileceği klasör seçilir.
  - Kaydedilecek dosya uzantısı seçilir. Nesne tespit işlemlerinde .jpeg ya da .jpg uzantılı dosyaların kullanılması gerektiği için burada JPEG seçilir.
  - Oluşan yeni resim dosyasının ölçülerinin ne olması gerektiği yazılır.
- Run butonuna basarak resim işleme adımı uygulanır.

Şekil 32: Resim Boyutlarının Düzenlenmesi



Kaynak: Yazar tarafından derlenmiştir.

4.2.4.2. Python Kodu ile Resim Boyutlarının Düzenlenmesi:

Modelimizin girdi olarak kullanacağı resimlerin standart boyutlarda olması, katmanlar arasında filtreleme işlemlerinden geçirilirken veri kaybının engellenmesi ve 3492 x 3492 px boyutundaki resimlerin eğitilirken eğitim süresini uzatabileceği düşünüldüğünde eğitim süresinin kısaltılması için resim boyutları 600 x 600 px değerlerine indirgenmiştir. Hatalı ve hatasız resimler için kullanılan Python kodu Ek-2'de belirtilmiştir.

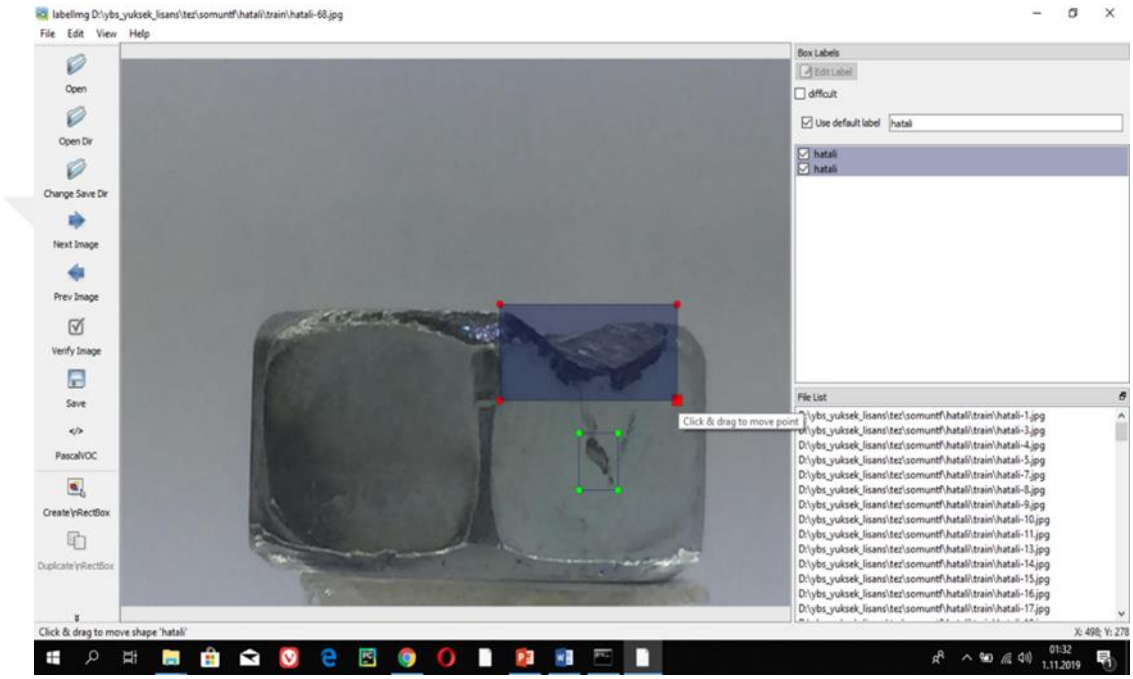
4.2.5. Resimlerin Etiketlenmesi

Gerçekleştirdiğimiz sistemin hatalı ve hatasız metal somunları tespit ederek onları sınıflandırması için "hatalı" ve "hatasız" olmak üzere iki adet sınıf oluşturulmuştur. Tespit ve sınıflandırma işleminin başarılı bir şekilde gerçekleştirilebilmesi için modelimizin bunu öğrenecek şekilde eğitilmesi

gerekmektedir. Bu da hatalı ve hatasız metal somun resimlerinin doğru olarak etiketlenerek modele verilmesi ile mümkün olmaktadır. Etiketleme işlemi aşağıda verilen Şekil 33'te görüldüğü gibi yapılmıştır.

4.2.5.1. LabelImg Resim Etiketleme Uygulaması

Şekil 33: LabelImg ile Bir Etiketleme İşlemi



Kaynak: Yazar tarafından derlenmiştir.

LabelImg, grafik resim etiketleme aracıdır. Python ile yazılmış ve grafik arayüz olarak Qt kullanır.

Etiketlenen resimlere ait etiket bilgileri, ImageNet tarafından kullanılan PASCAL VOC formatında, .xml dosyaları olarak kaydedilir. Bunun haricinde, etiket bilgileri YOLO formatında desteklemektedir.

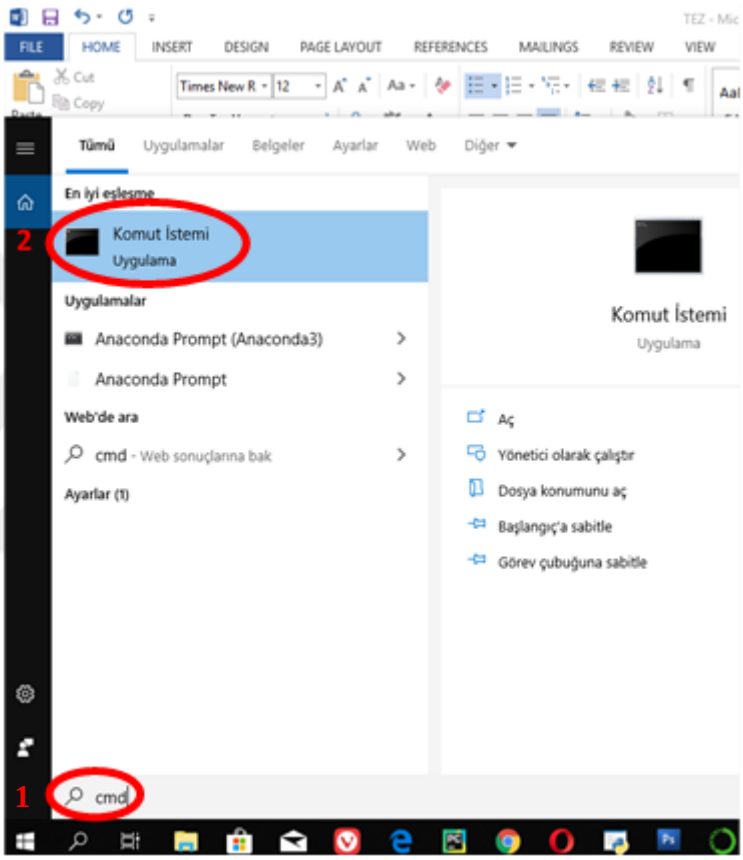
Biz etiketleme işlemini PASCAL VOC formatında yaptığımız için bu adımlara göre ilerleyeceğiz.

Windows işletim sistemine sahip bir laptop üzerinde etiketleme işlemi gerçekleştirildiği için Windows üzerinde uygulanan kurulum ve etiketleme adımları aşağıdaki gibi uygulanmıştır.

4.2.5.1.1. LabelImg Kurulumu

- LabelImg dosyaları indirilerek, indirilen zip dosyası kök dizine (C:\) açılır.
- Aşağıdaki Şekil 34'te belirtildiği gibi cmd komutu ile komut satırı penceresi açılır.

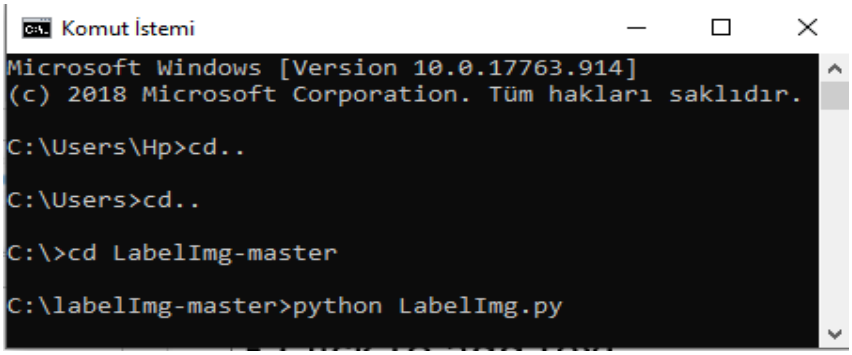
Şekil 34: Komut Satırının Açılması



Kaynak: Yazar tarafından derlenmiştir.

- Kök dizin içerisinde (C:\) C:\>cd LabelImg-master komutuyla etiketleme aracımızın setup klasörüne girilir.
- C:\LabelImg-master> dizininde C:\LabelImg-master>python LabelImg.py komutu ile uygulamamız açılır.

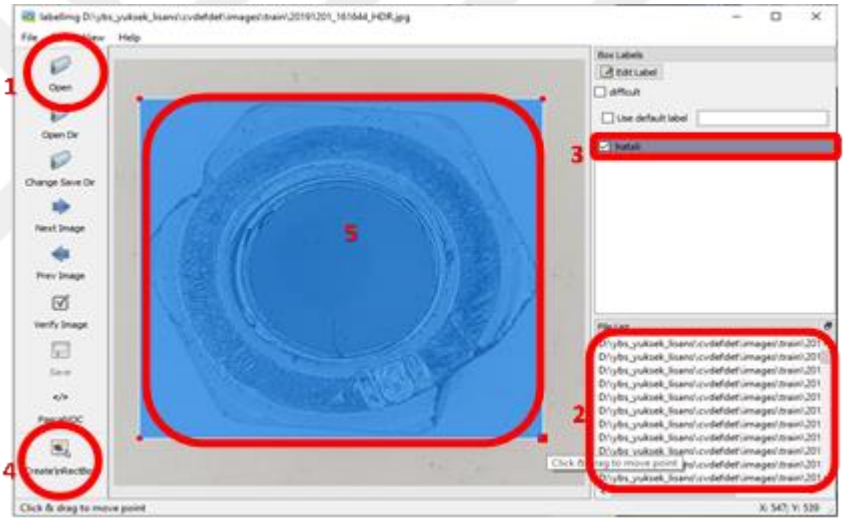
Şekil 35: Komut İstemi Üzerinden LabelImg Uygulaması Açılması



Kaynak: Yazar tarafından derlenmiştir.

4.2.5.1.2. Etiketleme

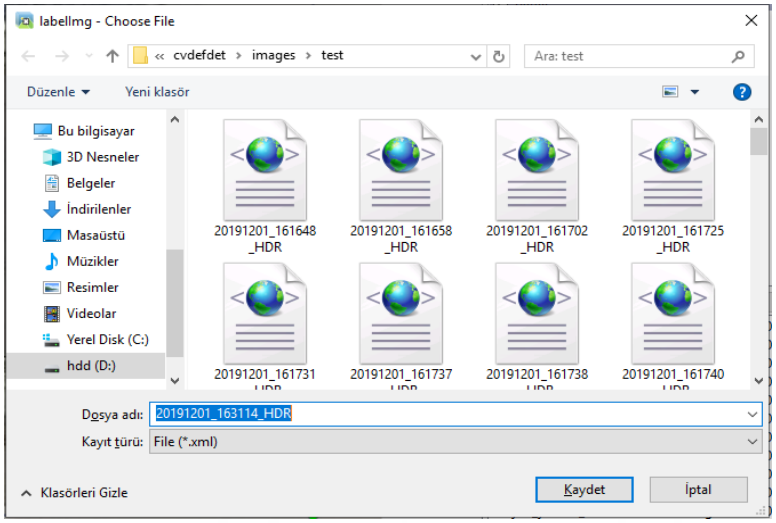
Şekil 36: LabelImg ile Etiketleme Örneği



Kaynak: Yazar tarafından derlenmiştir.

- Açılan pencerede “Open Dir” menüsü seçilerek etiketleme yapılması istenen resimlerin bulunduğu klasör seçilir ve resimler getirilir
- Etiket ismi belirtilir.
- Crete RectBox menüsü seçilir.
- Fare tuşuna basılı tutarak etiketlenmesi istenen alan sürükleyerek seçilir.
- Ctrl+S ile koordinatları içeren etiket bilgilerini saklayan .xml uzantılı dosya oluşturulmuş olur.

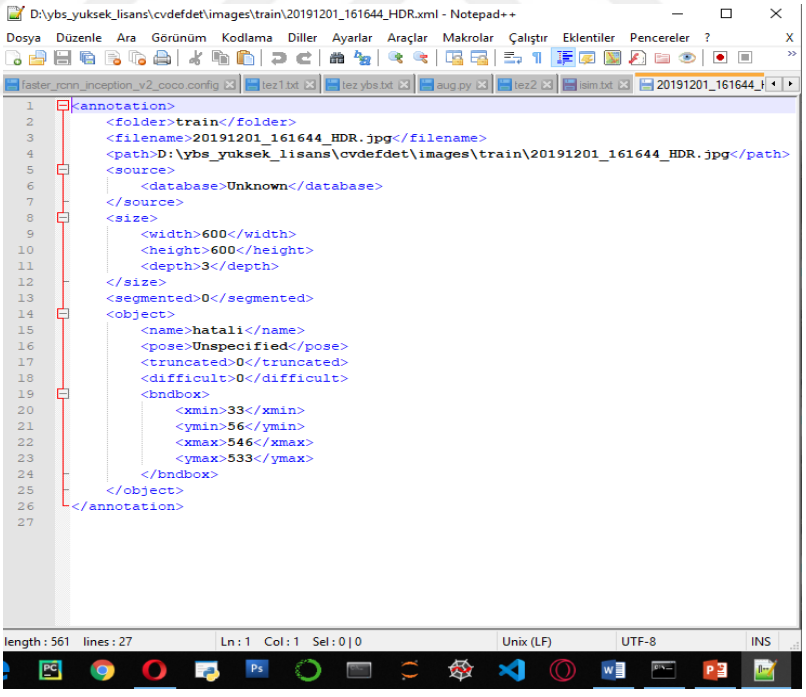
Şekil 37: Oluşturulan XML Dosyaları



Kaynak: Yazar tarafından derlenmiştir.

- Oluşan .xml dosyasının içeriği aşağıdaki şekilde gösterilmektedir.

Şekil 38: Xml Dosyasının İçeriği



Kaynak: Yazar tarafından derlenmiştir.

LabelImg Kısayol Tuşları Tablo 4'te gösterildiği gibidir:

Tablo 4: LabelImg Kısayol Tuşları

Kısayol	İşlevi
Ctrl + u	Resimlerin tamamını klasörden yükler
Ctrl + r	Varsayılan etiket hedef dosyasını değiştirir
Ctrl + s	Kaydet
Ctrl + d	Mevcut etiket ve kare kutuyu kopyalar
Space	Geçerli görüntüyü doğrulanmış olarak işaretler
w	Bir kare kutu oluşturur
d	Sonraki resme geçiş yapar
a	Önceki resme geçiş yapar
del	Seçili kare kutuyu siler
Ctrl++	Yakınlaştırır
Ctrl--	Uzaklaştırır
↑→↓←	Klavye okları seçili olan kare kutuyu hareket ettirir

Kaynak: Yazar tarafından derlenmiştir.

Verify Image (Resim Doğrulama), boşluğa basıldığında, kullanıcı görüntüyü doğrulandığı gibi işaretleyebilir, yeşil bir arka plan görünecektir. Bu, otomatik olarak veri kümesi oluştururken kullanılır, kullanıcı daha sonra tüm resimlerden sonra bunları ek açıklama yerine işaretleyebilir.

Difficult (Zor) ise, 1 olarak ayarlanır, nesnenin "zor" olarak açıklandığı, örneğin bağlamı büyük ölçüde kullanmadan açıkça görülebilen ancak tanınması zor bir nesne olarak belirlendiği belirtilir. Derin sinir ağı uygulamanıza göre, eğitim sırasında zor nesneler dahil edilebilir veya hariç tutulabilir. (Lin, 2015).

Etiketleme işlemi tamamlandıktan sonra derin öğrenme işlemini gerçekleştirmek için tensorflow nesne tespit API (Object Detection API) kullanılmıştır.

4.3. EĞİTİMDE KULLANILAN ARAÇLAR

Metal somun hata tespit sisteminin geliştirilmesi esnasında, derin öğrenme modelimizin eğitiminin yapılması için ihtiyaç duyulan donanım gereksinimleri Google'ın bulut hizmeti olarak sunmuş olduğu Colaboratory hizmeti kullanılmıştır. Yoğun işlem gücü gerektiren modelin eğitim aşaması, Google Colab'in sunmuş olduğu GPU ile hızlı bir şekilde gerçekleştirilmiştir.

Eğitim için TensorFlow kütüphanesinin Object Detection API'si kullanılarak, bu API içerisinde önceden hazırlanmış ve eğitilmiş olan başarılı modeller kullanılarak transfer öğrenmesi ve hazırladığımız veri seti ile bu modellerin yeniden eğitilmesi sağlanmıştır.

Eğitim aşamasında elde edilen verilerin görselleştirilmesi için yine Google Colab üzerinde çalıştırılabilen tensorboard platformu kullanılmıştır. Tensorboard üzerinden, eğitimin loss değerleri takip edilerek eğitimin başarılı olup olmadığına karar verilmiştir.

4.3.1. Google Colaboratory

Colaboratory, kurulum gerektirmeyen ve tamamen bulutta çalışan ücretsiz bir Jupyter dizüstü bilgisayar ortamıdır.

Colaboratory ile tarayıcınızdan ücretsiz olarak kod yazabilir ve yürütebilir, analizlerinizi kaydedebilir ve paylaşabilir ve güçlü bilgi işlem kaynaklarına erişebilirsiniz.

Google Colaboratory çalışma sayfaları (notebooks), Google Drive üzerinde açık kaynak Jupyter Notebook formatında yani .ipynb formatında depolanır. Yapılan çalışmalar, Google Drive dosya paylaşım adımları kullanılarak paylaşılabilir.

Colaboratory Python 2.7 ve Python 3.6 versiyonlarını destekler. Ancak 2020 yılından itibaren 2.7 desteği kesilecektir.

Colab ile Python programlama dilinde uygulama geliştirilebilir.

Keras, TensorFlow, PyTorch ve OpenCV gibi kütüphaneleri kullanılarak derin öğrenme (deep learning) uygulamaları geliştirilebilir.

Colab'ı,, ücretsiz hizmet veren diğer bulut hizmetlerinden ayıran en önemli özellik; GPU kullanımını ücretsiz olarak sağlamasıdır.

Hizmet verdiği Tesla K80 GPU ile derin öğrenme uygulamalarında modelin eğitilmesi için oldukça hızlıdır. Çalışma zamanı 12 saat sonunda sonlanmaktadır ve yeniden giriş yapılarak işlemlere devam edilebilmektedir. Bitcoin madenciliği uygulamalarını desteklememektedir.

4.3.2. Tensorboard

Tensorboard Tensorflow Geliştirme Ekibinin 2017 yılında ortaya attığı bir görselleştirme aracıdır.

Tensorboard ile eğitim parametreleri, metrikleri, hiper parametreler ya da sinir ağının herhangi bir istatistiği görselleştirilebilmektedir. Bu sayede yapılan anlık takip ile modele doğrudan anında müdahale yapma imkanı elde edilebilmektedir.

Histogram olarak veri koleksiyonu ya da vektörleri görselleştirilebilmektedir.

Öğrenme süresince sinir ağının ağırlık güncellemelerinin sinir ağlarına olan etkilerinin ağırlık yüklemeleriyle nasıl karşılaştırıldığı görülür.

Aşağıdaki başlıklarda görselleştirme işlemleri takip edilebilir:

- Scalars ile, sınıflandırma doğruluğu gibi skaler değerleri görselleştirir,
- Graph ile, sinir ağı modeli gibi, modelin grafik hesaplama değeri görselleştirilir.
- Distributions ile, sinir ağı ağırlıklarının zaman içerisinde nasıl değişim gösterdiği görselleştirilir.
- Histogramlar ile dağılımları 3 boyutlu perspektifte gösteren daha ince bir görünüm elde edilir.
- Projektör ile, Kelime gömülmelerini görselleştirmek için kullanılabilir (kelime gömülmeleri, onların anlamsal ilişkilerini yakalayan kelimelerin sayısal temsilidir)
- Görüntü ile, Görüntü verileri görselleştirilir.
- Ses ile, Ses verileri görselleştirilir.
- Metin ile, Metin (dize) verileri görselleştirilir.

4.3.3. Tensorflow Object Detection API

Tensorflow Nesne Algılama API'si, transfer öğrenme yöntemini kullanarak önceden eğitilmiş nesne algılama modellerinin kullanılmasını veya yeni modeller oluşturulmasını ve eğitilmesini sağlayan açık kaynaklı bir çerçevedir. Son derece kullanışlıdır, çünkü sıfırdan bir nesne algılama modeli oluşturmak zor olabilir ve çok fazla bilgi işlem gücü alabilir.

API, iNaturalist Tür Tespit Veri Kümesinde 4 milyon yineleme için eğitilmiş ResNet-50 ve ResNet-101 özellik çıkarıcılarını kullanarak nesneleri algılar.

Tensorflow Object Detection API, tensorflow1.x versiyonları ile oldukça başarılı uygulamalar geliştirilmesine yardımcı olmuş ve tensorflow2.0 ile birlikte yapısal bazı değişikliklere gitmiştir. Modüllerde yapılan güncellemeler ile birlikte 1.x versiyonlarında kullanılan bazı metotların kullanımı sona erdirilmiştir.

Çalışmamızda bu konuda sorun yaşanmış ve Google Colab üzerinde yapılan eğitim esnasında ve nesne tespit modülünün çalıştırılması esnasında çözümü mümkün olmayan ya da uzun süren oldukça fazla sayıda hata ile karşılaşılmıştır.

Modelimiz eğitilirken Tensorflow Nesne Algılama API kullanılmış ve ModelZoo üzerinden ulaşılan, nesne tespit için kullanılan MS COCO veri seti ile eğitilmiş olan Faster RCNN Inception V2 modeli üzerinden transfer öğrenmesi gerçekleştirilerek eğitim işlemi tamamlanmıştır.

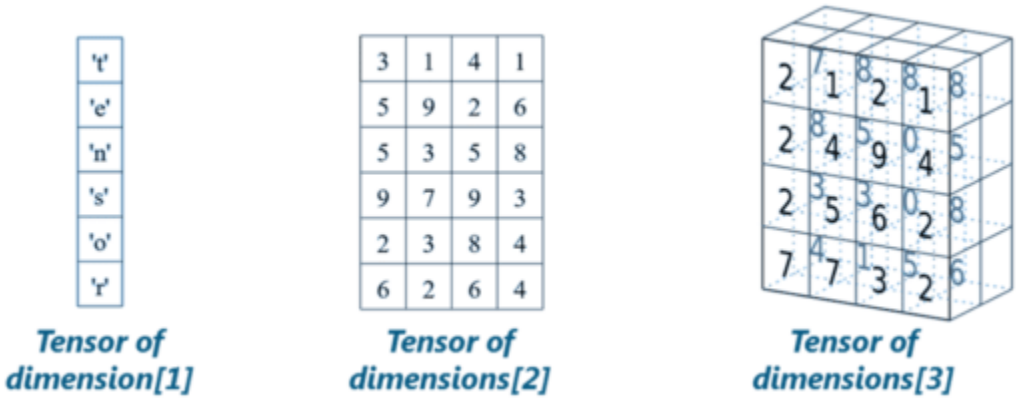
Şekil 39: Tensorflow Logosu

Kaynak: Tensorflow, 2015

4.3.3.1. Tensör

Tensör, derin öğrenmede verileri temsil eden soyut bir kavramdır, fiili bir kavram değildir.

Şekil 40: Tensör Dizileri



Kaynak: Bakshi, 2019

Derin öğrenmede, yüksek boyutlu veri kümeleriyle uğraşılır. Şekil 40'da görüleceği üzere, tensörler yalnızca yüksek boyutlu olan verilerin temsil edilmesine izin veren çok boyutlu dizilerdir ve herhangi bir boyutta olan bir dizi içinde tutulan ilkel (integer, float) verilerden oluşan veri setidir. "TensorFlow" ismi, sinir ağlarının tensörler üzerinde gerçekleştirdiği işlemlerden ortaya çıkmıştır..

4.3.3.2. Graph

TensorFlow'da tüm işlemler bir grafik (graph) içerisinde gerçekleştirilir. Graph art arda gerçekleşen bir hesaplama kümesidir. Graph, bir eğitim sırasında yapılan tüm hesaplamaları tanımlar. Graph'ın bir çok avantajı vardır:

Birden fazla CPU veya GPU üzerinde ve hatta mobil işletim sisteminde çalışmak için geliştirilmiştir.

Graph'ın taşınabilirliği, derhal veya daha sonra kullanım için hesaplamaları korumaya izin verir. Graph gelecekte çalıştırılmak üzere kaydedilebilir.

Graph'ta bulunan tüm hesaplamalar, tensörleri birbirine bağlayarak yapılır.

Bir tensörün bir düğümü ve bir kenarı vardır. Düğüm matematiksel işlemi taşır ve bir uç nokta çıktısı üretir. Kenarlar, düğümler arasındaki giriş / çıkış ilişkilerini açıklar (Bakshi, A., 2019).

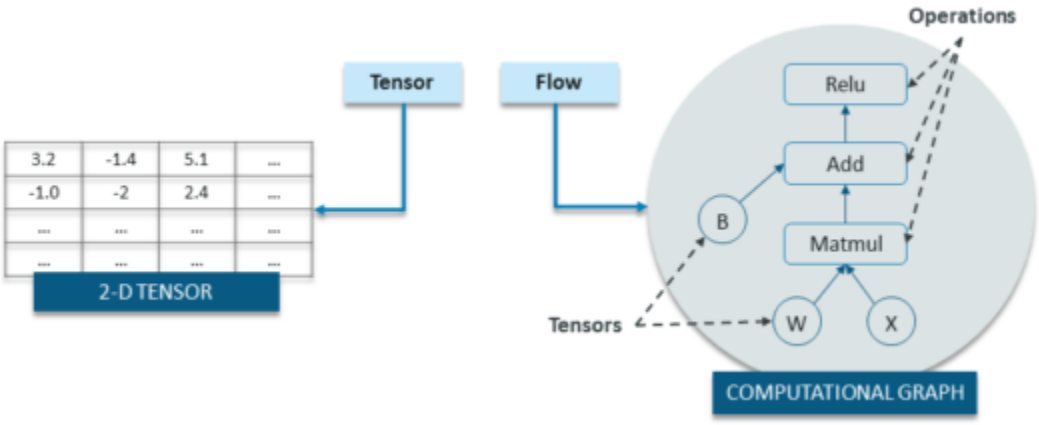
4.3.3.3. Tensorflow

Tensorflow, Google tarafından, yeni başlayanların ve uzmanların makine öğrenimi modelleri oluşturmasını kolaylaştıran, makine öğrenmesi uygulamaları için geliştirilmiş uçtan uca açık kaynaklı bir kütüphanedir (https://www.tensorflow.org/overview/).

TensorFlow terimi iki terimden oluşur Tensor & Flow.

TensorFlow'da tensör (tensor) terimi, verilerin çok boyutlu dizi olarak temsil edilmesine karşılık gelirken, akış (flow)terimi, Şekil 41'de gösterildiği gibi bir tensör üzerinde gerçekleştirilen işlem serilerini ifade eder.

Şekil 41: TensorFlow Yapısı

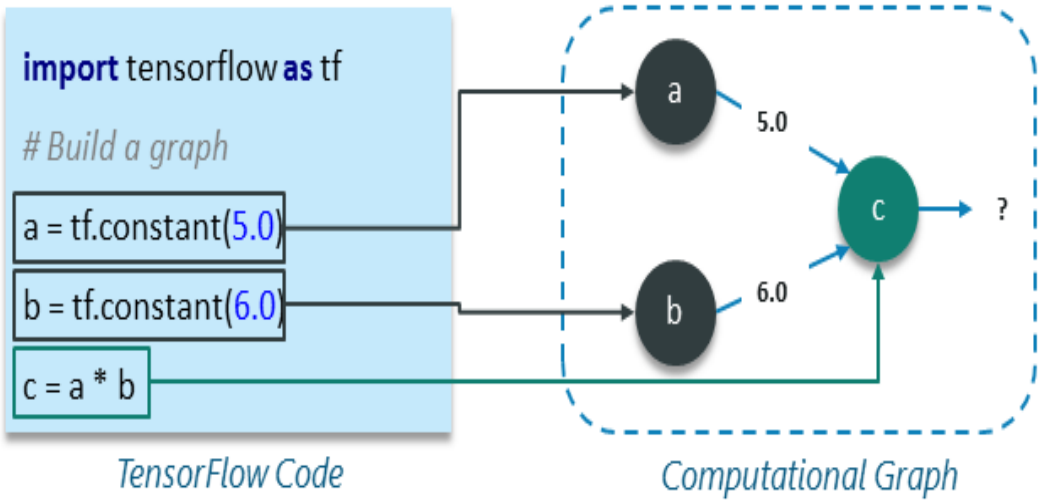


Kaynak: Bakshi, 2019

Bir tensorflow programı yazma işlemi genellikle iki adımdan oluşur:

- Hesaplamalı grafik oluşturmak: Bir hesaplama grafiği, grafikte düğümler olarak düzenlenmiş bir dizi TensorFlow işlemini ifade eder. Her düğüm, giriş olarak 0 veya daha fazla tensörü alır ve çıkış olarak bir tensör üretir. Üç düğümden oluşan (a, b ve c) örnek bir hesaplama grafiği Şekil 42'de görülmektedir.

Şekil 42: Tensorflow Computational Graph



Kaynak: Bakshi, 2019

Sabit düğümler, sabit değerleri sıfır giriş olarak saklamak için kullanılır, ancak kayıtlı değerleri çıktı olarak üretir. Şekil 42'deki örnekte a ve b, sırasıyla 5 ve 6 değerlerine sahip sabit düğümlerdir. c düğümü, sabit a düğümünün b düğümü ile çarpılması işlemini temsil eder. Bu nedenle, c düğümünün yürütülmesi, a ve b sabit düğümlerinin çoğaltılmasına neden olur.

Bir hesaplama grafiği, TensorFlow programında yer alan matematiksel hesaplamaları kavramsallaştırmanın alternatif bir yolu olarak düşünülebilir. Hesaplamalı grafiğin farklı düğümlerine atanan işlemler paralel olarak gerçekleştirilebilir, böylece hesaplamalar açısından daha iyi bir performans sağlanır.

- Hesaplamalı grafik çalıştırmak: c düğümünün çıktısını almak için, hesaplama grafiğini bir oturumda çalıştırmamız gerekiyor. Oturum, grafik işlemlerini CPU veya GPU gibi aygıtlara yerleştirir ve bunları yürütmek için yöntemler sağlar.

Bir oturum, TensorFlow çalışma zamanının kontrolünü ve durumunu kapsüllemekte, yani, tüm işlemlerin gerçekleştirileceği sırayla ilgili bilgileri saklamakta ve önceden hesaplanmış işlemin sonucunu akış hattındaki bir sonraki işleme geçirmektedir. Hesaplama grafiğinin bir oturumda çalıştıran kod Şekil 43'te gösterilmiştir.

Şekil 43: Hesaplama Grafiğinin Çalıştırılması

```
# Create the session object
sess = tf.Session()

#Run the graph within a session and store the output to a variable
output_c = sess.run(c)

#Print the output of node c
print(output_c)

#Close the session to free up some resources
sess.close()
```

```
Output:
30
```

Kaynak: Bakshi, 2019

4.4. TENSORFLOW OBJECT DETECTION API İLE EĞİTİMİN GERÇEKLEŞTİRİLMESİ

Tensorflow, görüntüleri otomatik olarak tespit etmek için CNN gibi sinir ağı modelleri oluşturmamıza yardımcı olur. Bu kapsamda Tensorflow görüntü tanıma için iki yaklaşım ortaya koyar;

- Sınıflandırma: CNN'i kedi, köpek, araba vb. gibi nesne kategorilerini tanıyacak şekilde eğitilir. Eğitilen sistem bu kategorilere göre görüntüyü bir bütün olarak sınıflandırır.
- Nesne Algılama: sınıflandırmadan daha güçlüdür ve aynı görüntü içinde birçok nesneyi algılar. Ayrıca algıladığı nesneleri etiketlerve görüntü üzerinde nesnelerin yerlerini gösterir.

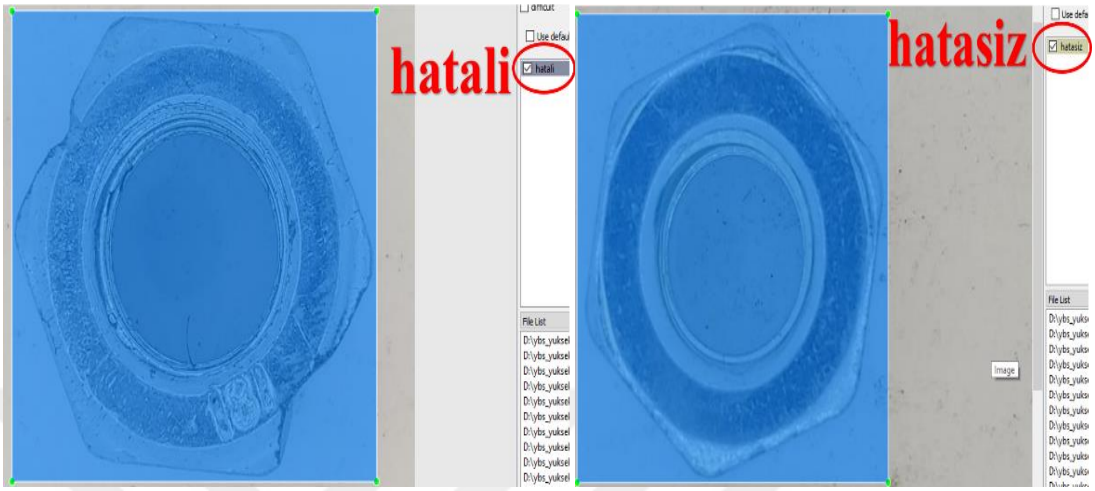
Bu çalışmada, her iki adım da uygulanarak metal somunların hata tespit ve sınıflandırma işlemleri gerçekleştirilmiştir.

4.4.1. Veri Seti İşlemleri

Tez dokümanımızın "4.2. VERİLERİN HAZIRLANMASI" başlığı kapsamında veri seti hazırlanmış, veri artırım işlemleri uygulanmış, yeniden boyutlandırma gerçekleştirilmiş, 2000 adet resimden oluşan veri setindeki resimler

%80 train (1600) ve %20 test (400) verisi olacak şekilde ayrılmış ve resimlerin "hatalı" ve "hatasız" olarak etiketleme işlemi Şekil 44'te görüldüğü gibi tamamlanmıştır.

Şekil 44: Etiketleme Örnekleri



Kaynak: Yazar tarafından derlenmiştir.

4.4.2. Object Detection API Kurulumu ve Derin Öğrenme

Bu aşamada Tensorflow Object Detection API kurulumu yapılmış, kullandığımız model olan Faster R-CNN Inception V2 modelimiz yüklenmiş, train ve test klasörlerinde yer alan etiket bilgileri öncelikle CSV formatına ve ardından da CSV formatından tensorflow'un anlayacağı tfRecord dosyasına dönüştürülmüştür.

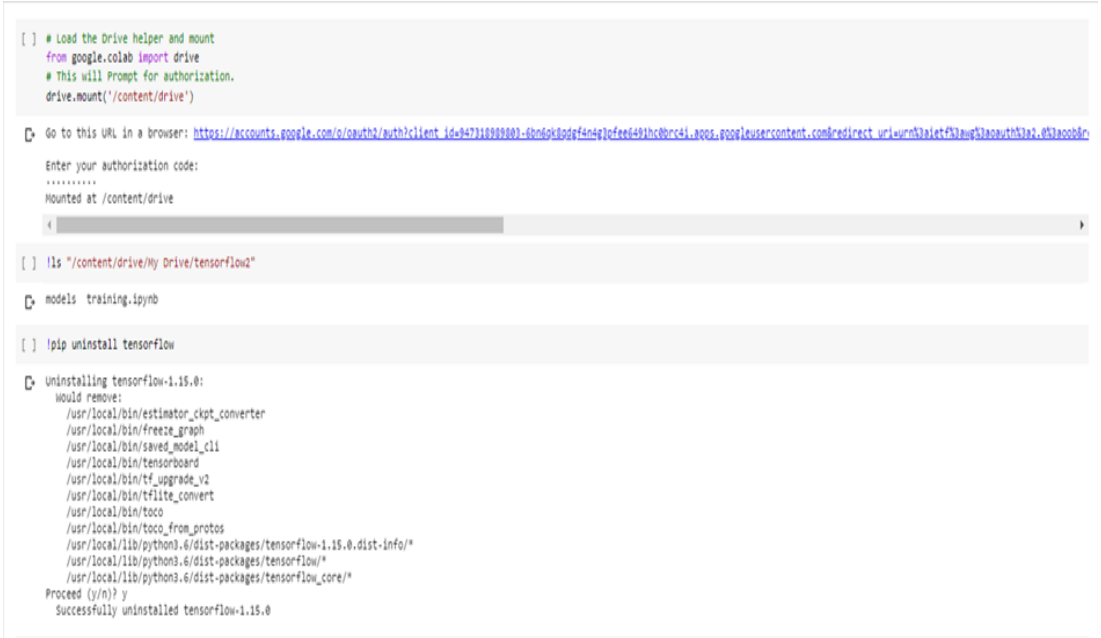
Model konfigürasyonumuz modele ait config dosyası içerisinde yapılmış, etiket atamalarının yapıldığı label.pbxt dosyası yapılandırılmış ve transfer öğrenmesiyle birlikte eğitim işlemi bize ait olan veri setiyle yeniden yapılmıştır.

Inference Graph'la birlikte modelimiz en uygun eğitim adımında dondurularak kullanıma hazır hale getirilmiştir.

Bu aşamada aşağıdaki adımlar sırasıyla uygulanmıştır:

- Google Colaboratory kullanıcı giriş işlemleri Şekil 45'te olduğu gibi yapılmıştır. Gelen linke tıklanarak elde edilen kullanıcı yetki anahtarı girilerek Colab üzerindeki klasörümüze giriş yapılmıştır.

Şekil 45: Google Colaboratory Dizinine Giriş



Kaynak: Yazar tarafından derlenmiştir.

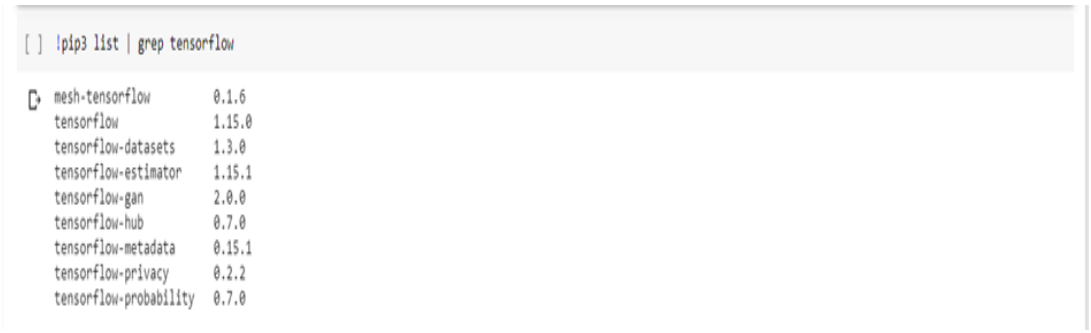
- Şekil 46'da görülen adım ile gerekli olan kütüphaneler yüklenmiştir. Burada tensorflow 2.0 güncellemesiyle yapılan değişikliklerden meydana gelen hatalardan kaçınmak ve GPU üzerinde işlem yapabilmek için tensorflow-gpu= = 1.13.2 versiyonu kullanılmıştır.

Şekil 46: Kütüphanelerin Yüklenmesi

Kaynak: Yazar tarafından derlenmiştir.

- Şekil 47'de belirtilen kodun çalıştırılmasıyla yüklenen kütüphaneler kontrol edilmiştir.

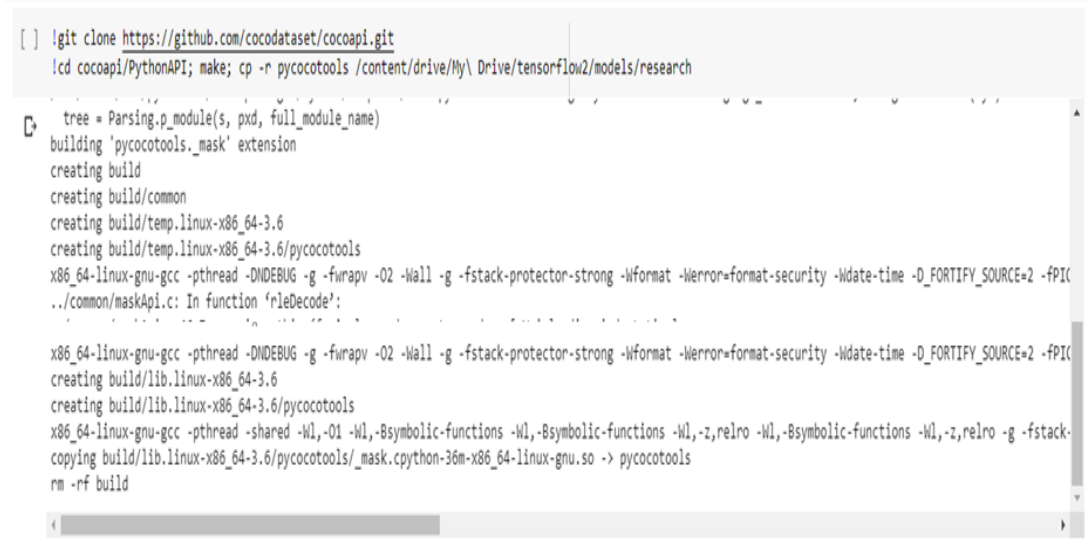
Şekil 47: Kütüphanelerin Kontrol Edilmesi



Kaynak: Yazar tarafından derlenmiştir.

- Şekil 48'de gösterilen yöntemle birlikte MS COCO modelimiz uygulamamıza yüklenmiştir.

Şekil 48: MS COCO Modelinin Yüklenmesi



Kaynak: Yazar tarafından derlenmiştir.

- Şekil 49 ile modelimiz inşa edilmiştir.

Şekil 49: Modelin İnşa Edilmesi

Kaynak: Yazar tarafından derlenmiştir.

- Şekil 50 ile, hazırlanan modelimiz yüklenmiştir.

Şekil 50: Modelin Yüklenmesi

Kaynak: Yazar tarafından derlenmiştir.

- Şekil 51 ile modelimiz test edilmiştir.

Şekil 51: Modelin Test Edilmesi

Kaynak: Yazar tarafından derlenmiştir.

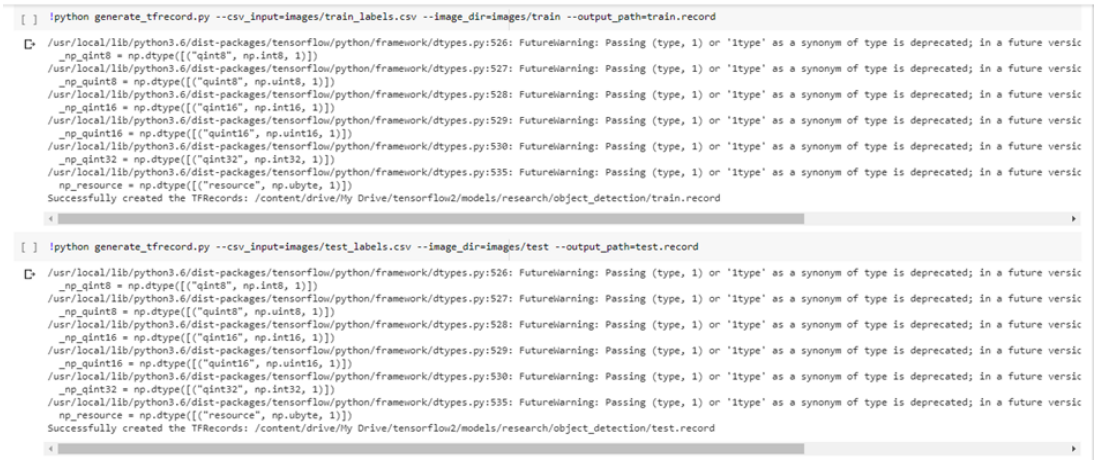
- Şekil 52'de görüldüğü gibi object detection klasörüne girilerek bu dosya içerisinde yer alan xml dosyaları csv formatına dönüştürülür.

Şekil 52: Etiket Bilgilerinin XML-CSV Dönüşümü

Kaynak: Yazar tarafından derlenmiştir.

- Şekil 53'te belirtilen işlem ile CSV formatına dönüştürülmüş olan resim etiket verileri tensorflow record dosyalarına dönüştürülür.

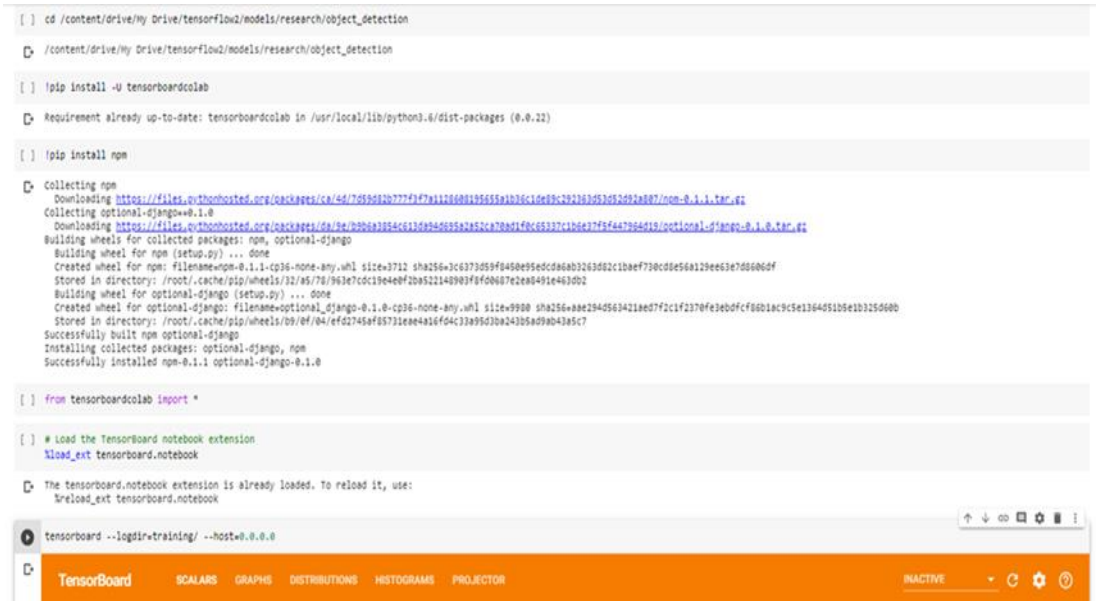
Şekil 53: CSV Dosyalarının tfRecord Dosyalarına Dönüştürülmesi



Kaynak: Yazar tarafından derlenmiştir.

- Şekil 54' de belirtilen kod dizisiyle birlikte Tensorboard kurulumu ve veri geldiğinde görselleştirme işlemini yapabilmesi için 0.0.0.0 host'u üzerinden sürekli olarak dinleme durumuna geçilmiştir.

Şekil 54: Tensorboard Kurulumu ve Dinlemeye Hazır Hale Getirilmesi



Kaynak: Yazar tarafından derlenmiştir.

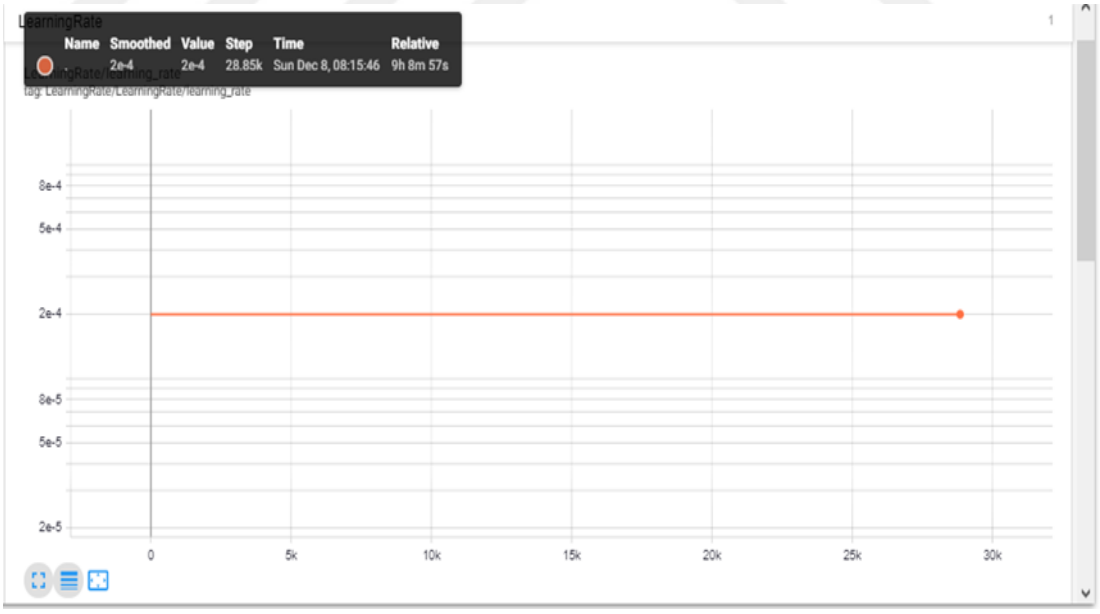
- Şekil 55'te eğitim işleminin gerçekleştirilmesi görülmektedir.

Şekil 55: Eğitimin Gerçekleştirilmesi

Kaynak: Yazar tarafından derlenmiştir.

- Eğitimin başlamasıyla birlikte Tensorboard'ta verileri okumaya başlamıştır. Aldığı verileri görselleştirerek grafik olarak okunabilir hale getirmiştir. Şekil 56 Modelimizin Learning Rate grafiğini göstermektedir.

Şekil 56: Learning Rate Grafiği



Kaynak: Yazar tarafından derlenmiştir.

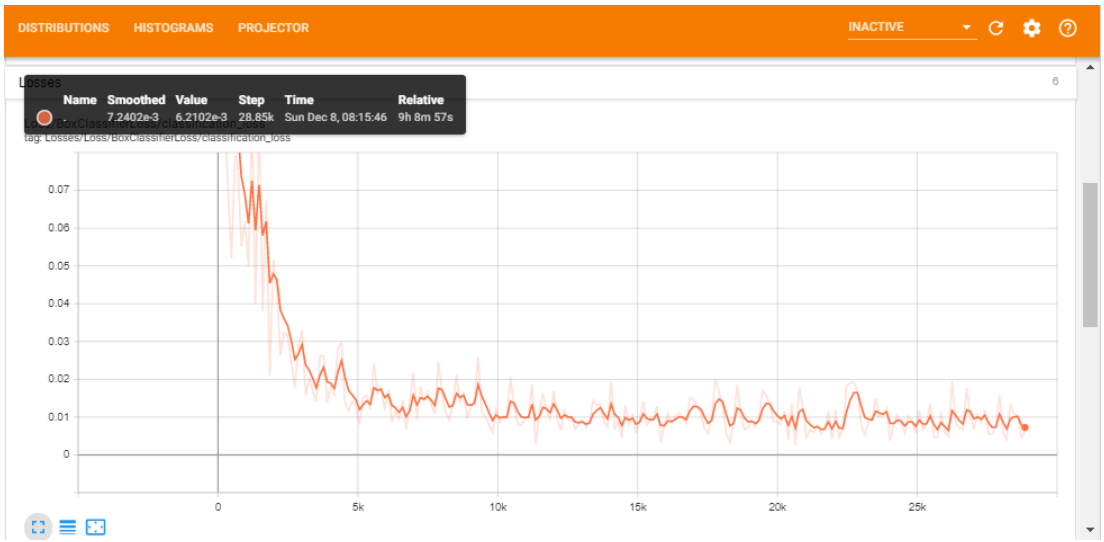
- Şekil 57**,** Loss grafiklerini göstermektedir.

Şekil 57: Loss Grafiklerinin Genel Görünümü

Kaynak: Yazar tarafından derlenmiştir.

- Şekil 58'de BoxClassifierLoss/classification_loss grafiği görülmektedir. Bu grafik, tespit edilen nesnelerin çeşitli sınıflara sınıflandırılması aşamasında takip edilen loss değerini gösterir.

Şekil 58: Loss/BoxClassifierLoss/classification_loss



Kaynak: Yazar tarafından derlenmiştir.

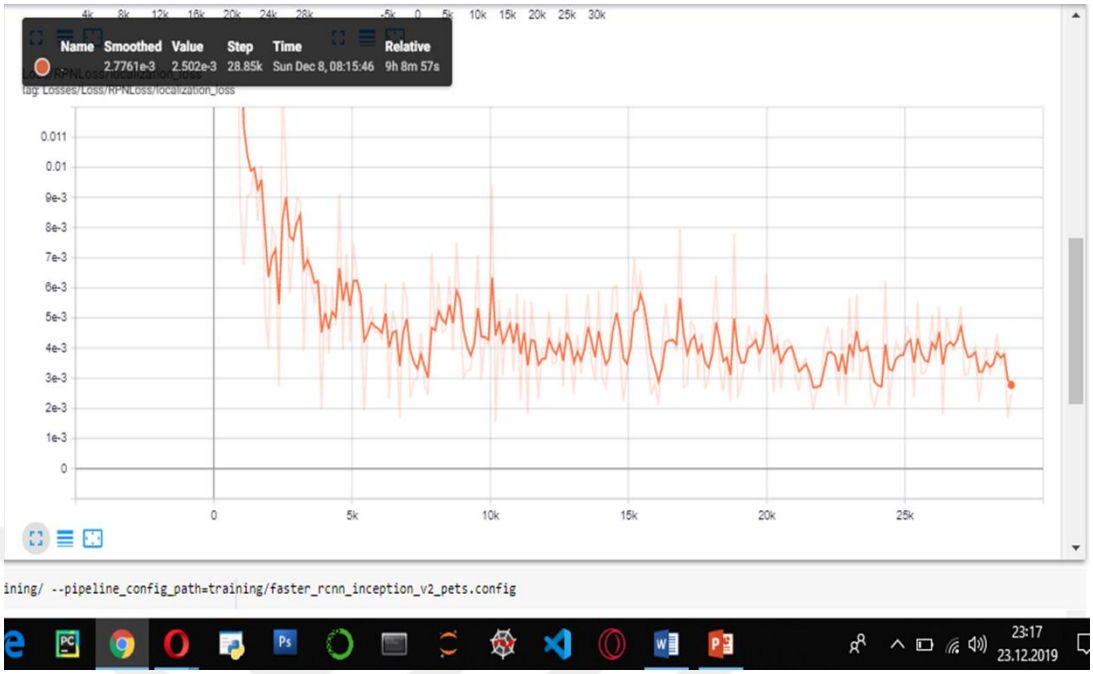
- Şekil 59, BoxClassifierLoss/localization_loss grafiğini gösterir. Bu grafik, Yerelleştirme Kaybı veya Sınırlayıcı Kutu regresörünün Kaybını görselleştirmektedir.

Şekil 59: Loss/BoxClassifierLoss/localization_loss

Kaynak: Yazar tarafından derlenmiştir.

- Şekil 60, RPNLoss/localization_loss grafiğini gösterir. Bu kayıp RPN için Yerelleştirme Kaybı veya Sınırlayıcı Kutu regresörünün kaybını göstermektedir.

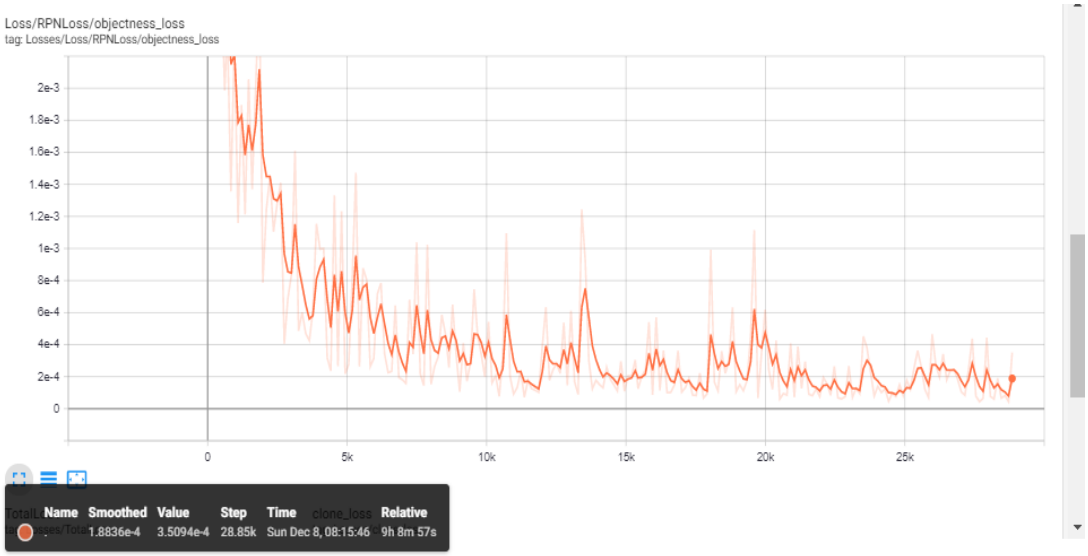
Şekil 60: Loss/RPNLoss/localization_loss



Kaynak: Yazar tarafından derlenmiştir.

- Şekil 61, Sınırlayıcı bir kutunun ilgi alanı veya arka plan nesnesi olup olmadığını sınıflandıran sınıflandırıcı'nın kaybını gösteren grafiği içerir.

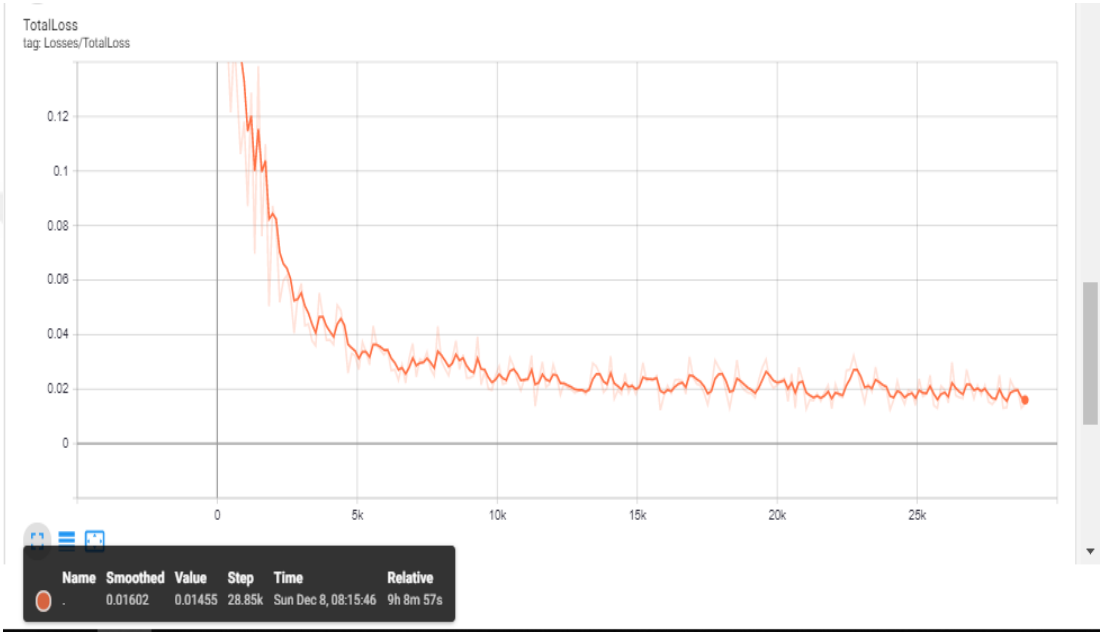
Şekil 61: Loss/RPNLoss/objectness_loss



Kaynak: Yazar tarafından derlenmiştir.

- Şekil 62 Total Loss değerini göstermektedir. Bu değer modelimiz için takip edilmesi gereken en önemli loss grafiğidir. Bu grafiği takip ederek modelin en uygun noktasında eğitimi durdurma ve dondurma ya da bir hata durumunda eğitimi durdurup hatayı düzeltme gibi işlemleri yaparak zaman kazanmamızı sağlar.

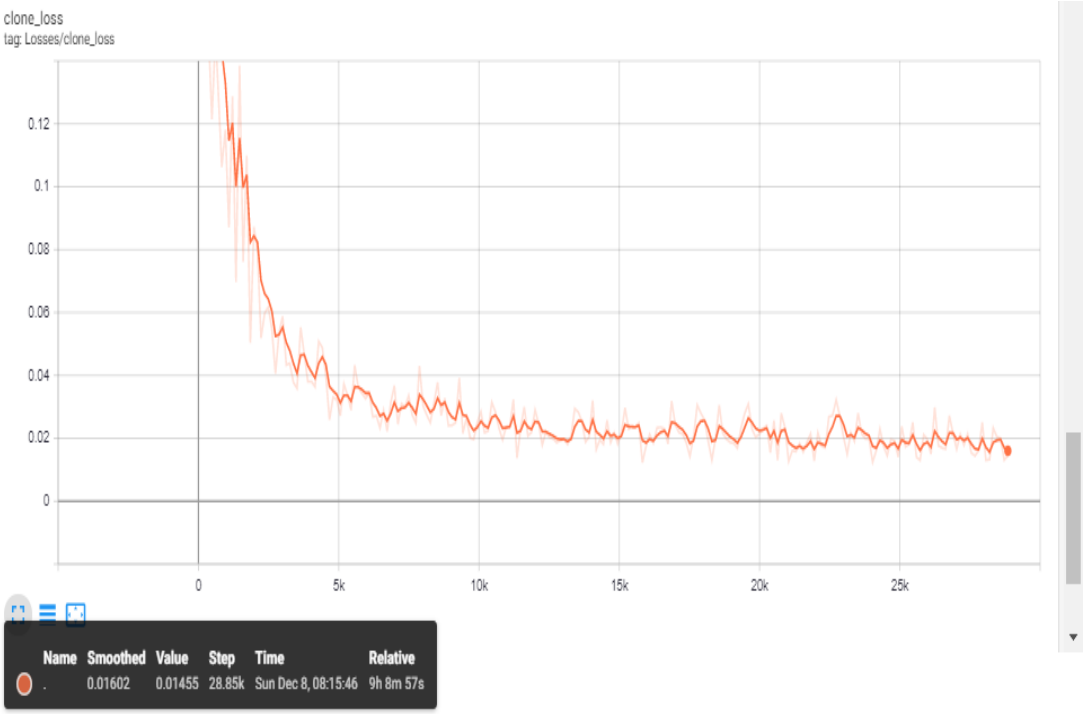
Şekil 62: Total Loss



Kaynak: Yazar tarafından derlenmiştir.

- Şekil 63 Clone Loss grafiğini göstermektedir. Clone Loss değeri eğer birden fazla GPU kullanılırsa bir anlam ifade etmektedir. clone_loss_1 yalnızca birden fazla GPU üzerinde eğitim veriyorsanız geçerlidir: Tensorflow, her GPU üzerinde eğitim almak ve her bir klondaki kaybı bildirmek için modelin bir kopyasını oluşturur. Modeli tek bir GPU / CPU'da eğitiyorsanız, TotalLoss ile aynı olan clone_loss_1 öğesini görürsünüz. Biz eğitimimizde bir adet GPU kullandığımız için değerlerimiz Total Loss değerleriyle aynı olmuştur.

Şekil 63: Clone Loss

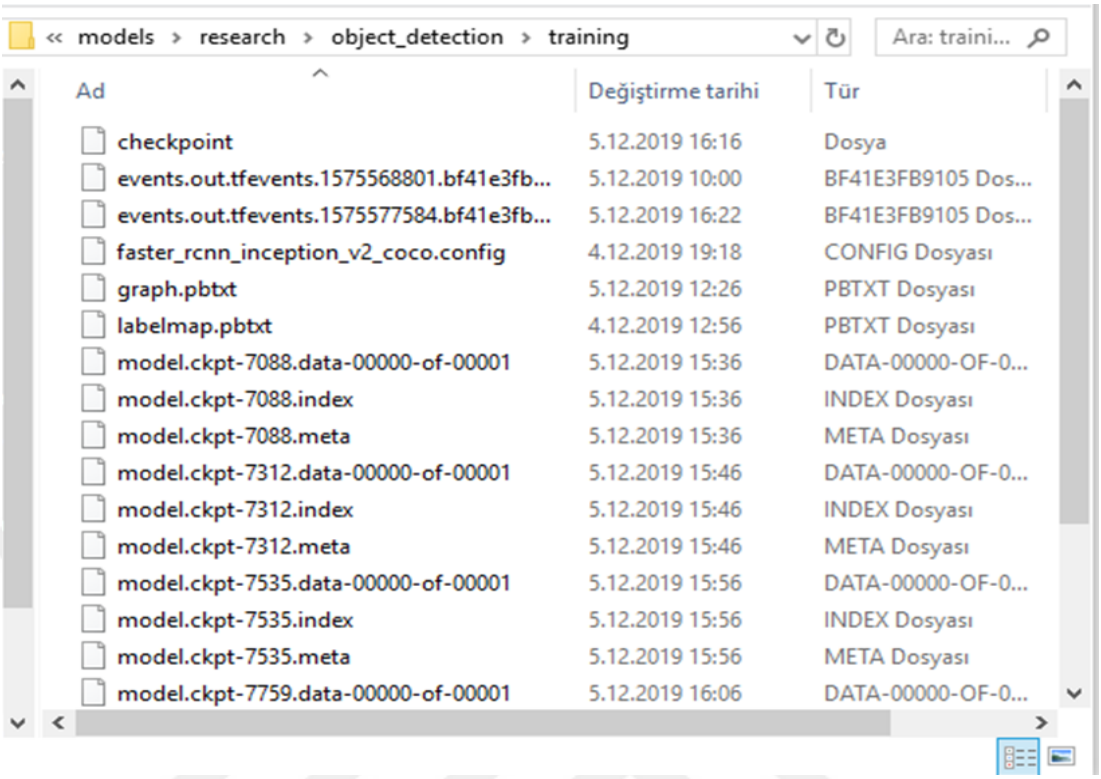


Kaynak: Yazar tarafından derlenmiştir.

Eğitimimiz 28.850'nci adıma kadar gerçekleştirilmiştir. COCO modeli ile eğitim yapıldığı için loss değeri 0,05 ve altında bir değere kadar istikrarlı bir şekilde düşmesi gerekmektedir. Biz 28.850 adım eğittikten sonra loss değerimiz 0,01 değerlerinde seyretmeye başlamıştır. Loss değeri istediğimiz çerçevede olduğu için modelin ezberlemesini (overfitting) engellemek amacıyla eğitimimiz 28.850'nci adımda sonlandırılmıştır.

- Eğitim esnasında periyodik olarak yedeklenen eğitim verileri training klasörü içerisine otomatik olarak kaydedilir.

Şekil 64: training Klasörünün İçeriği



Kaynak: Yazar tarafından derlenmiştir.

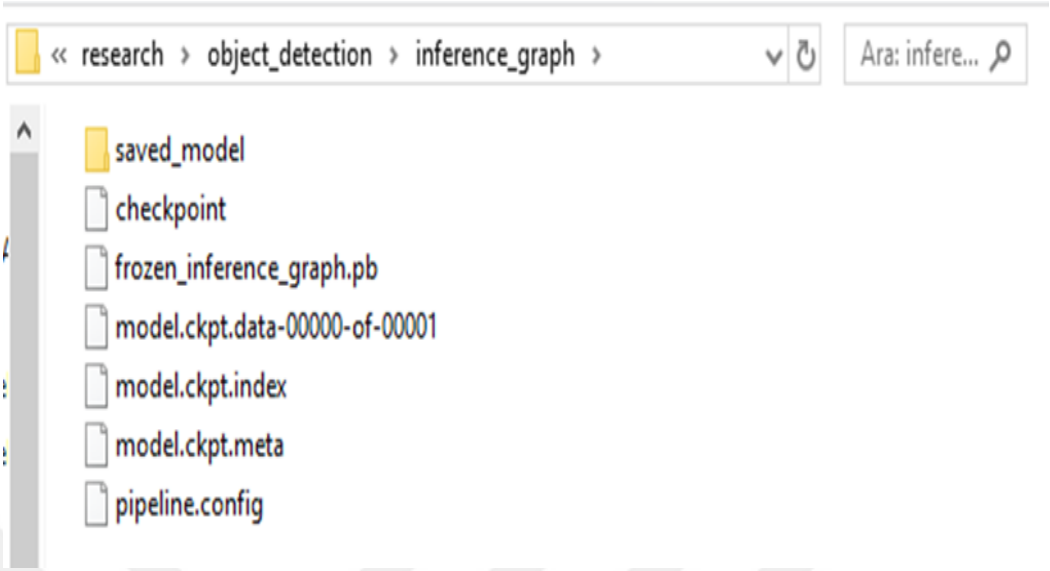
- Eğitimin tamamlanmasının ardından elde edilen verilerin hata tespit ve sınıflandırma işleminde kullanılması için dondurulması gereklidir. Dondurma işlemi aşağıda belirtilen Şekil 65'te gösterilen kod vasıtasıyla yapılır.

Şekil 65: Inference Graph ile Eğitim Verilerinin Dondurulması

Kaynak: Yazar tarafından derlenmiştir.

Dondurma işleminden sonra Object_detection klasöründe Inferece_graph klasörü oluşur ve 28.350'nci adımda dondurulan eğitim verileri, hata tespit ve sınıflandırma işleminde kullanılmak üzere buraya kaydedilir. Şekil 66'da inference_graph klasörünün içeriği gösterilmiştir.

Şekil 66 : Inference Graph Klasörü



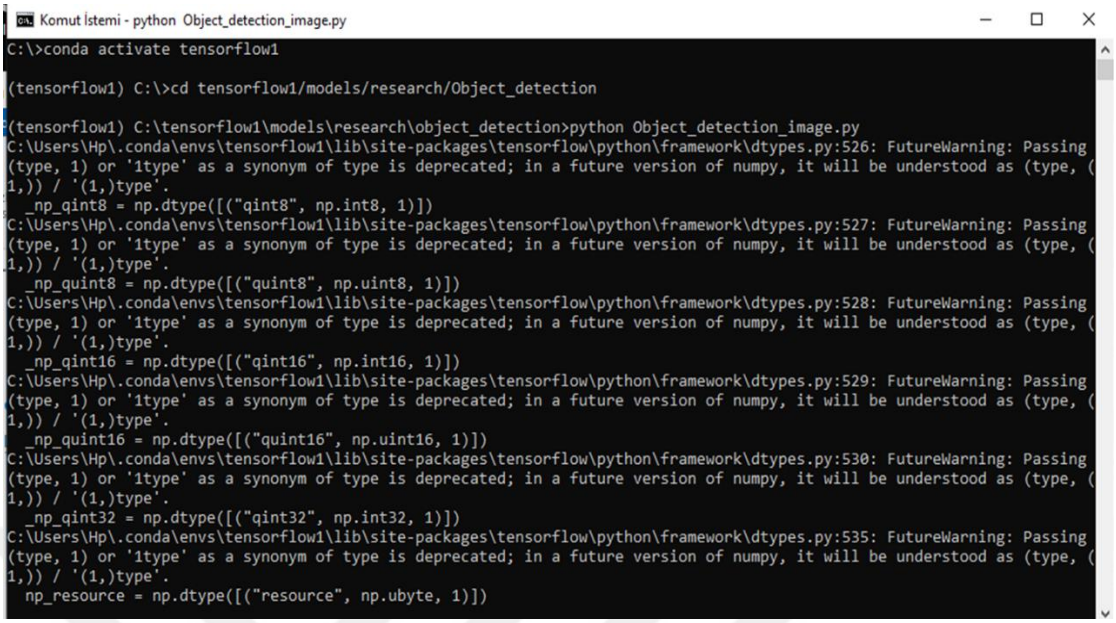
Kaynak: Yazar tarafından derlenmiştir.

4.4.3. Bilgisayarla Görü İşlemi

Eğitimimiz başarılı bir şekilde gerçekleştikten sonra artık hata tespit ve sınıflandırma işlemi yapan bilgisayarla görü adımı uygulanabilir hale gelmiştir.

Ama önce tensorflow 2.0 güncellemesinden kaynaklanan uyumsuzluk sorunundan dolayı Google Drive üzerinde bulunan dosyaların tamamı yerel bilgisayarın C:\ sürücüsüne tensorflow1 adıyla kaydedilmiştir. Ve bilgisayarla görü işlemi bu klasör içerisinde yer alan Object_detection_image.py ve Object_detection_webcam.py modülleri komut penceresi üzerinden çalıştırılarak gerçekleştirilmiştir.

Şekil 67: Komut İstemi Üzerinde Object_detection _image.py Modülü Çalıştırılması



Kaynak: Yazar tarafından derlenmiştir.

- Object_detection_image.py modülü çalıştırıldıktan sonra, yerel dosyada yer alan test resimleri üzerinde hata tespit ve sınıflandırma işlemi gerçekleştirilmiş, ardından Object_detection_webcam.py modülü çalıştırılarak gerçek zamanlı görüntüler üzerinde hatalı ve hatasız metal somunlar üzerinde tespit ve sınıflandırma işlemi gerçekleştirilmiştir.

Şekil 68: Hatasız Metal Somunun Resim Üzerinde %99 Doğrulukla Tespit ve Sınıflandırılması

Kaynak: Yazar tarafından derlenmiştir.

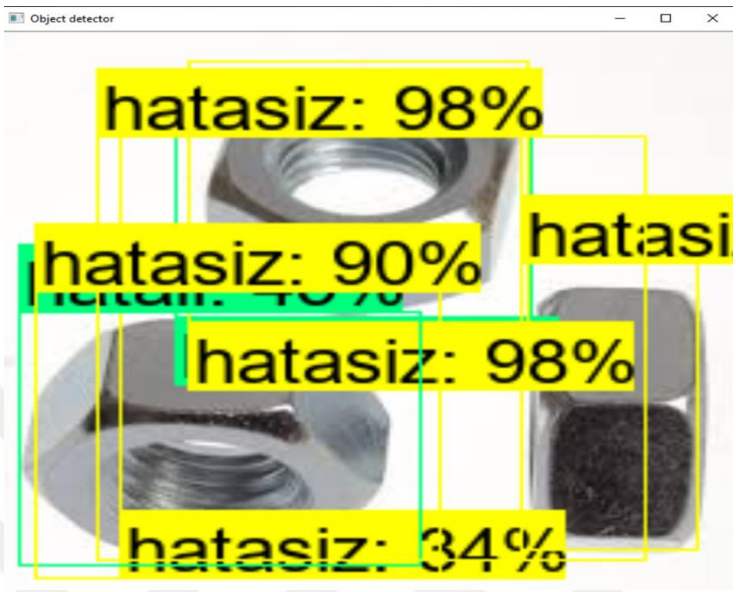
Şekil 69: Hatalı Metal Somunun Resim Üzerinde %99 Doğrulukla Tespit ve Sınıflandırılması

Kaynak: Yazar tarafından derlenmiştir.

Yukarıda Şekil 68 ve Şekil 69'da verilen resimlerden de görüleceği üzere gerçekleştirilen sistem ile bir tek metal somunun bulunduğu ve uygun ortam koşullarında veri setine uygun açılardan alınan görüntülerde nesne tespit işleminin

kolayca yapıldığı ve aynı zamanda da %99 gibi oranlarda doğru sınıflandırma işleminin yapıldığı anlaşılmaktadır.

Şekil 70: Birden Fazla Nesne Hata Tespit ve Sınıflandırılması

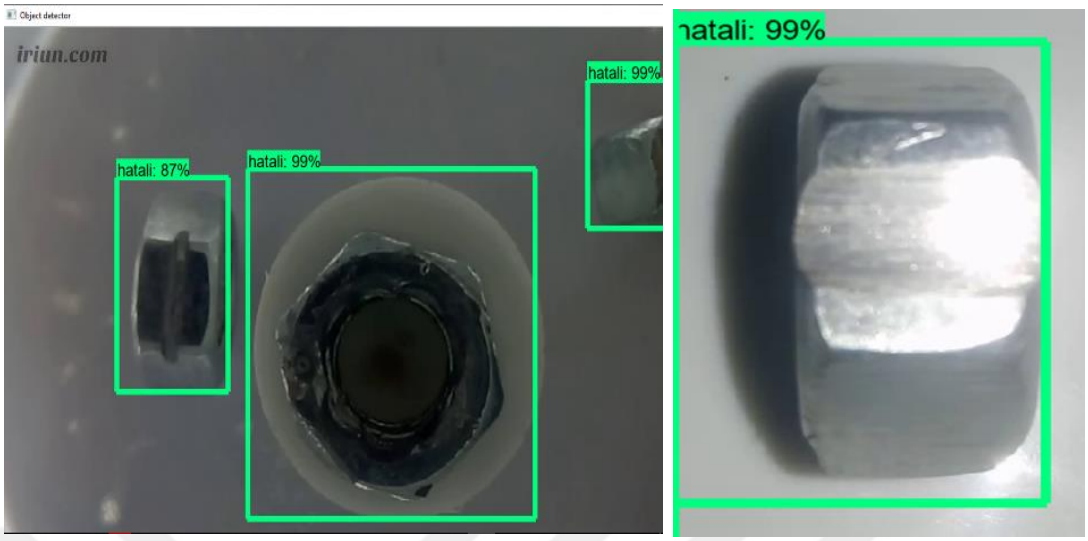


Kaynak: Yazar tarafından derlenmiştir.

Şekil 70'te görüldüğü gibi, farklı açılardan birden fazla nesnenin yer aldığı görüntü üzerinde yapılan işlemde ise veri setine uygun açılarda bulunan somunların tespit işleminin yapıldığı ve sınıflandırma oranının da %90'lar seviyesinde bir doğrulukta olduğu anlaşılmaktadır. Ancak, uyumsuz açılardan gelen görüntülerde nesne tespit işleminde sorun yaşanmadığı, bunun yanında sınıflandırma işleminde hatalar meydana geldiği belirlenmiştir.

Veri setimizdeki resimlerin tamamı üstten ve yandan olmak üzere yalnızca iki açıdan alınan görüntüler ile oluşturulduğu için resim üzerinde yapılan hata tespit ve sınıflandırma uygulamasında üst ve yan açılardan alınan metal somun görüntülerinin tespit ve sınıflandırma doğruluğu %99 seviyelerine kadar çıkmaktadır.

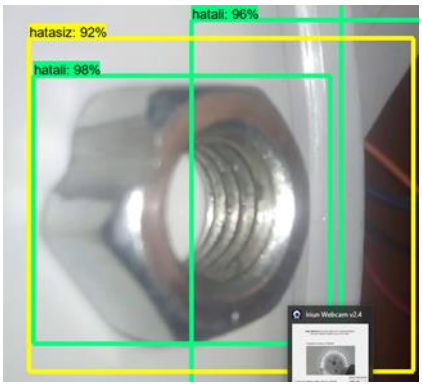
Şekil 71: Uygun Koşullarda Nesne Tespit İşlemi Örnekleri



Kaynak: Yazar tarafından derlenmiştir.

Veri setinde belirlenen üst ve yan açılardaki görüntüler haricinde farklı açılardan gelen görüntüler üzerinde de RPN (bölge öneri ağı) sınırlandırma kutuları (bounding box) oluşturarak tespit ve lokalizasyon gerçekleştirmeye çalışmaktadır. Bu nedenle resimdeki metal somunların farklı bölgelerinde hatalı ya da hatasız etiketleme işlemi gerçekleştiği görülmektedir.

Şekil 72: RPN'in Farklı Açılardan Bounding Box Oluşturma İsteği



Kaynak: Yazar tarafından derlenmiştir.

Şekil 73: Hatalı Sınıflandırma ve Bounding Box Örneği

Kaynak: Yazar tarafından derlenmiştir.

Benzer bir işlem de somun resimlerinin webcam üzerinden alınan gerçek zamanlı görüntüler üzerinde de meydana gelmiştir. Dik veya yan metal somun görüntülerinde, aydınlatma yeterli ise %90 - %99 aralığında bir doğrulukla hata tespit ve sınıflandırma işleminin gerçekleştiği, ancak ışığın yetersiz olduğu durumlarda sistemin doğruluk oranının düştüğü, özellikle yan açılardan gelen metal somun görüntülerinde yetersiz aydınlatmadan dolayı hatalı sınıflandırmalar meydana geldiği anlaşılmıştır. Ayrıca, aynı metal somunların arka zemin renginde değişim meydana geldiğinde, görüntünün farklı bölgelerinde de bounding boxların görülmeye başlandığı belirlenmiştir.

SONUÇ VE ÖNERİLER

Gerçekleştirilen uygulama ile, önerilen sistemin çalışmanın başında belirtilen amaçlarına uygun olarak endüstriyel alanda sağlayacağı çözümlere yönelik umut verici sonuçlar alındığı düşünülmektedir.

Operasyonel açıdan bakıldığında, uygulama bilgisayarla görü vasıtasıyla insan faktörünün yerini alarak, akıllı bir otomasyon süreci oluşturmuş, kalite kontrol işleminin minimum hata ile sürdürülmesine yardımcı olmuş, hataya neden olan sürecin tespit edilmesine katkıda bulunarak süreç iyileştirmelerinin belirlenmesinde etkin rol oynayacağı görülmüştür.

Yapılan denemelerde, üst ve yan taraflarından elde edilen görüntülerde yer alan metal somunların duruş açılarının önemli olduğu görülmüştür. Veri setine uygun olmayan açılarda alınan görüntülerde, bölge öneri ağının sınırlandırma kutusu oluşturma isteğinden dolayı farklı açılarda birden çok sınırlandırma kutusu oluşturduğu ve sınıflandırma işleminde başarının düştüğü görülmüştür.

Derin öğrenme yöntemi ile metal somun resimlerine ait üst ve yan açılardan çekilmiş resimlerden oluşan veri setine uygulanan eğitimler ile webcam üzerinden gerçek zamanlı olarak elde edilen ya da yerel bilgisayarda bulunan test resimleri üzerinden alınan görüntüler ile yapılan tespit ve hata sınıflandırma işlemlerinde, veri setine uygun aydınlatma ve açılarda gerçekleştirilen işlem, %90-%99 oranları arasında tespit ve sınıflandırma doğruluğuyla başarılı sonuçlar vermiştir.

Derin öğrenme ve bilgisayarla görü çözümlerine duyulan ihtiyaç ve ilginin artarak devam ettiği günümüzde, bilgisayarla görü uygulamalarının üretim ve denetim süreçlerine entegre edilmesinin kaçınılmaz olduğu açıkça görülmektedir.

Önerilen yöntemin bir sınırlaması, derin bir ağın eğitiminin, çok zaman ve masraf gerektiren, elle etiketlenmiş verileri gerektirmesidir. Geliştirilen sistem şu anda prototip aşamasındadır. Tez kapsamında geliştirilen uygulamanın endüstriyel alana katkısının süreklilik arz edebilmesi için, gelecek çalışmalarda,

- Veri setindeki veri sayısının artırılması,
- Verilerin tamamı üzerinde veri artırım işleminin uygulanması,
- Mask RCNN kullanılarak hata tiplerine göre sınıflandırmanın yapılması,

- Kapsül ağlarının kullanılarak Faster RCNN in gerektirdiği açısal görünüm zorluklarından kurtularak, herhangi bir açıda doğru tespit ve sınıflandırma avantajı sağlanması,
- Uygulamanın Raspberry Pi gibi mini bilgisayarlar üzerinde çalışabilir hale getirilmesi,
- Birden fazla kameralar üzerinden elde edilen eş zamanlı görüntüler kullanılması,
- Kendi kendine öğrenen bir tasarım uygulanması
- Kullanıcıya ikaz ve öneri sunan bir yazılımın entegre edilmesi gibi önemli geliştirmelerle başarım oranının artırılması, sistem kararlılığının ve kullanılabilirliğinin artırılması sağlanabilir.

KAYNAKÇA

Alamsyah, D. ve Fachrurrozi, M. (2019). Faster R-CNN with Inception V2 for Fingertip Detection in Homogenous Background Image. *Journal of Physics Conference Series.* 1196(1): 2-3.

Alpaydın, E. (2011). *Yapay Öğrenme*. İstanbul: Boğaziçi Üniversitesi Yayınevi.

Aminanto, M.E. ve Kim, K. (2016). Deep Learning in Intrusion Detection System: An Overview. *Computer Science.* 2016: 4-6.

Aydın, İ.H. ve Değirmenci, C.H. (2018). *Yapay Zekâ*. İstanbul: Girdap Yayınevi

Ashton, K. (2009). That 'Internet of Things' Thing, RFID Journal. https://www.rfidjournal.com/articles/pdf?4986: 1, (26.12.2019).

Bakshi, A. (2019). TensorFlow Tutorial – Deep Learning Using TensorFlow. https://www.edureka.co/blog/tensorflow-tutorial/, (26.12.2019).

Beşer, F., Kızrak, M.A., Bolat, B. ve Yıldırım, T. (2018). Kapsül Ağları ile İşaret Dili Tanıma. *IEEE 26. Sinyal İşleme ve İletişim Uygulamaları (SİU) Konferansı,* Düzenleyen: İzmir Katip Çelebi Üniversitesi, İzmir. 2-5 Mayıs 2018.

Brownlee, J. (2019). How to Configure Image Data Augmentation in Keras. https://machinelearningmastery.com/how-to-configure-image-data-augmentation-when-training-deep-learning-neural-networks/, (26.12.2019).

Campos, M., Martins, T., Ferreira, M. ve Santos, C. (2008). Detection of Defects in Automotive Metal Components Through Computer Vision. *IEEE International Symposium on Industrial Electronics* (ss. 860-865), Düzenleyen Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE) Cambridge, Birleşik Krallık. 30 Haziran-2 Temmuz 2008.

Cebeci, H.İ. (2019). Bilgisayar Görüşü ile Yüz ve Nesne Tanıma | R-CNN, SSD, GANs. https://www.udemy.com/course/bilgisayar-gorusu/, (16.10.2019).

Cha, Y. J., Choi, W., Suh, G., Mahmoudkhani, S., ve Büyüköztürk, O. (2018). Autonomous Structural Visual Inspection using Regionbased Deep Learning for Detecting Multiple Damage Types, *Computer-Aided Civil and Infrastructure Engineering*. 33(9): 731-747.

Cirean, D.C., Meier, U., Masci, J. ve Gambardella, L. M. (2012). Flexible, High Performance Convolutional Neural Networks for Image Classification. *Twenty-Second International Joint Conference On Artificial Intelligence* (1237–1242), Barselona, İspanya. 19-22 Temmuz 2011.

Ferguson, M., Ak, R., Lee, Y. ve Law, K.H. (2018). Detection and Segmentation of Manufacturing Defects with Convolutional Neural Networks and Transfer Learning. *Smart and Sustainable Manufacturing Systems.* 2(1): 137-164.

Gao, Y. and Mosalam, K. M. (2018). Deep Transfer Learning for ImageBased Structural Damage Recognition. *Computer-Aided Civil and Infrastructure Engineering*. 33: 748–768.

Feng, C., Zhang, H., Wang, S., Li, Y., Wang, H., Yan, F. (2019). Structural Damage Detection using Deep Convolutional Neural Network and Transfer Learning. *KSCE Journal of Civil Engineering*. 23(10): 4493–4502.

Gonzalez, R.C., Woods, R.E. (1992). *Digital Image Processing*. New Jersey: Prentice Hall.

Gopalakrishnan, K., Khaitan: K., Choudhary, A. and Agrawal, A. (2017). Deep Convolutional Neural Networks with Transfer Learning For Computer Vision-Based

Data-Driven Pavement Distress Detection. *Construction and Building Materials.* 157: 322-330.

Deng, L. (2014). A Tutorial Survey Of Architectures, Algorithms and Applications for Deep Learning. *APSIPA Transactions on Signal and Information Processing.* 3(2014): 3-8.

Deng, L., Yu, D. ve diğerleri. (2014). Deep Learning: Methods and Applications. *Foundations and Trends R in Signal Processing*. 7(3–4): 197–387.

Gopalakrishnan, K., Khaitan, S.K., Choudhary, A. Ve Agrawal, A. (2017). Deep Convolutional Neural Networks with transfer learning for computer vision-based data-driven pavement distress detection. *Construction and Building Materials.* 157(1): 322-330.

Gopikrishna, Y. (2018). Object Detection Using TensorFlow and COCO Pre-Trained Models. https://medium.com/object-detection-using-tensorflow-and-coco-pre/object-detection-using-tensorflow-and-coco-pre-trained-models-5d8386019a8**,** (20.12.2019).

Hajizadeh, S., N´u˜nez, A., Tax, D.M.J. (2016). Semi-supervised Rail Defect Detection from Imbalanced Image Data. *IFAC (International Federation of Automatic Control),* 49(3): 78-83.

Hinton, G.E., Osindero, S., Teh, Y.W. (2006). A Fast Learning Algorithm For Deep Belief Nets. *Neural Computation.* 18(7): 1527-1554.

Kolar, Z., Chen, H. ve Luo, X. (2018). Transfer Learning and Deep Convolutional Neural Networks for Safety Guardrail Detection in 2d Images. *Automation in Construction.* 89: 58–70.

Krizhevsky, A., Sutskever, I. ve Hinton, G. E. (2012). Imagenet Classification with Deep Convolutional Neural Networks. *Advances in Neural Information Processing*

Systems 25 (NIPS 2012) (ss. 1097- 1105), Düzenleyen: Neural Information Processing Systems Foundation Inc., Tahoe Gölü, Amerika Birleşik Devletleri. 3-8 Aralık 2012.

LeCun, Y., Boser, B., Denker, J.S., Henderson, D., Howard, R.E., Hubbard, W. ve diğerleri. (1989). Backpropagation Applied to Handwritten Zip Code Recognition. *Neural Computation.* 1: 541-551.

LeCun, Y., Bottou, L., Bengio, Y. ve Haffner, P. (1998). Gradient-Based Learning Applied To Document Recognition. *Proceedings of the IEEE.* 86(11): 2278–2324.

LeCun, Y., Kavukcuoglu, K. ve Farabet, C. (2010). Convolutional Networks and Applications in Vision. *Proceedings of 2010 IEEE International Symposium on In Circuits and Systems* (ss. 253–256), Düzenleyen: Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE) Paris, Fransa. 30 Mayıs-2 Haziran 2010.

LeCun, Y., Bengio, Y. ve Hinton, G. (2015). Deep Learning. *Nature.* 521(2015): 436-444.

Lin, T. (2015). LabelImg. https://github.com/tzutalin/labelImg, (26.12.2019).

Manaswi, N.K. (2018). Convolutional Neural Networks. *Deep Learning With Applications Using Python* (ss. 91-96). Kaliforniya: Apress.

Mane, D. T. ve Kulkarni, U. V. (2019). A Survey on Supervised Convolutional Neural Network and Its Major Applications. *Deep Learning and Neural Networks: Concepts, Methodologies, Tools, and Applications* (ss. 1058-1071). Amerika Birleşik Devletleri: IGI Global.

Metaverbis (2019). Computer Vısıon And Image Processıng Solutions. http://metaverbis.com/computer-vision, (26.12.2019).

MissingLink (2016). Convolutional Neural Network Tutorial: From Basic to Advanced. https://missinglink.ai/guides/convolutional-neural-networks/convolutional-neural-network-tutorial-basic-advanced/, (25.12.2019).

Nielsen, M.A. (2015). Neural Networks and Deep Learning. http://neuralnetworksanddeeplearning.com/, (20.12.2019).

Patterson, J., Gibson, A. (2017). *Deep Learning*. Kaliforniya: O'Reilly Media, Inc.

Reis, Z.A. (2017). Yapay Zeka. *Mühendislikte Yapay Zeka Uygulamaları* (ss. 11-23). Sakarya: Sakarya Üniversitesi Kütüphanesi Yayınevi.

Robles, L.F., Azzopardi, G., Alegrea, E., Petkov, N. (2017). Machine-Vision-Based Identification of Broken Inserts in Edge Profile Milling Heads. *Robotics and Computer-Integrated Manufacturing*. 44(2017): 276-283.

Russ, J.C. (1995). *The Image Processing Handbook 2dh edition.* Florida: CRC Press.

Russakovsky, O., . Deng, j., · Su, H., · Krause, J., Satheesh, S., · Ma, S., · Huang, Z., Karpathy, A., Khosla, A., Bernstein, M., Berg, A.C., · Fei, L.F. (2015). ImageNet Large Scale Visual Recognition Challenge. *International Journal of Computer Vision.* 115(3): 211-252.

Russell, S.J., Norvig, P. (1995). *Artificial Intelligence a Modern Approach*. Contributing Writers: John F. Canny, Jitendra M. Malik, Douglas D. Edwards. New Jersey: Prentice Hall, Englewood Cliffs.

Russell, B.S. J., Norvig, P. (2009). Artificial Intelligence: A Modern Approach (3rd edition). New Jersey: Prentice Hall.

Saurabh, G. (2018). Use Machine Learning to Detect Defects on the Steel Surface. https://software.intel.com/en-us/articles/use-machine-learning-to-detect-defects-on-the-steel-surface, (24.12.2019).

Silver, D., Huang, A., Maddison, C. J., Guez, A., Sifre, L., Driessche, Schrittwieser, G. V. D. J., Antonoglou, I., Panneershelvam, V., Lanctot, M. ve diğerleri (2016), Mastering The Game of Go with Deep Neural Networks And Tree Search. *Nature*. 529(7587): 484–489.

Song, L., Li, X., Yang, Y., Zhu , X., Guo, Q. ve Yang, H. (2018). Detection of Micro-Defects on Metal Screw Surfaces Based on Deep Convolutional Neural Networks. *Sensors.* 18(11): 3709.

Staar, B., Lütjen, M., Freitag,M. (2019). Anomaly Detection with Convolutional Neural Networks for İndustrial Surface Inspection. *12th CIRP Conference on Intelligent Computation in Manufacturing Engineering* (ss. 484-489), Düzenleyen CIRP, Napoli Körfezi, İtalya. 18-20 Temmuz 2018.

Stackoverflow (2019). Module 'tensorflow' Has No Attribute 'contrib'. https://stackoverflow.com/questions/55870127/module-tensorflow-has-no-attribute-contrib, (26.12.2019).

Suetens, P., Fua, P., Hanson, A.J. (1992). Computational Strategies for Object Recognition. *ACM Computing Surveys*. 24(1): 5-61.

Szegedy, C., Vanhoucke, V., Ioffe: ve Shlens, J. (2016). Rethinking the Inception Architecture for Computer Vision. *IEEE conference on Computer Vision and Pattern Recognition* (ss. 2818-2826), Düzenleyen Elektrik ve Elektronik Mühendisleri Enstitüsü (IEEE), Las Vegas.

Tao, X., Zhang, D., Ma, W., Liu, X. ve Xu, D. (2018). Automatic Metallic Surface Defect Detection and Recognition with Convolutional Neural Networks. *Applied Science.* 8(9): 1575.

Tecim, V. (2015). Sistem Geliştirme Yaşam Döngüsü. http://debis.deu.edu.tr/userweb/vahap.tecim/dosyalar/sgyd.pdf , (19.12.2019).

Tensorflow (2017). TensorFlow Core. https://www.tensorflow.org/, (20.12.2019).

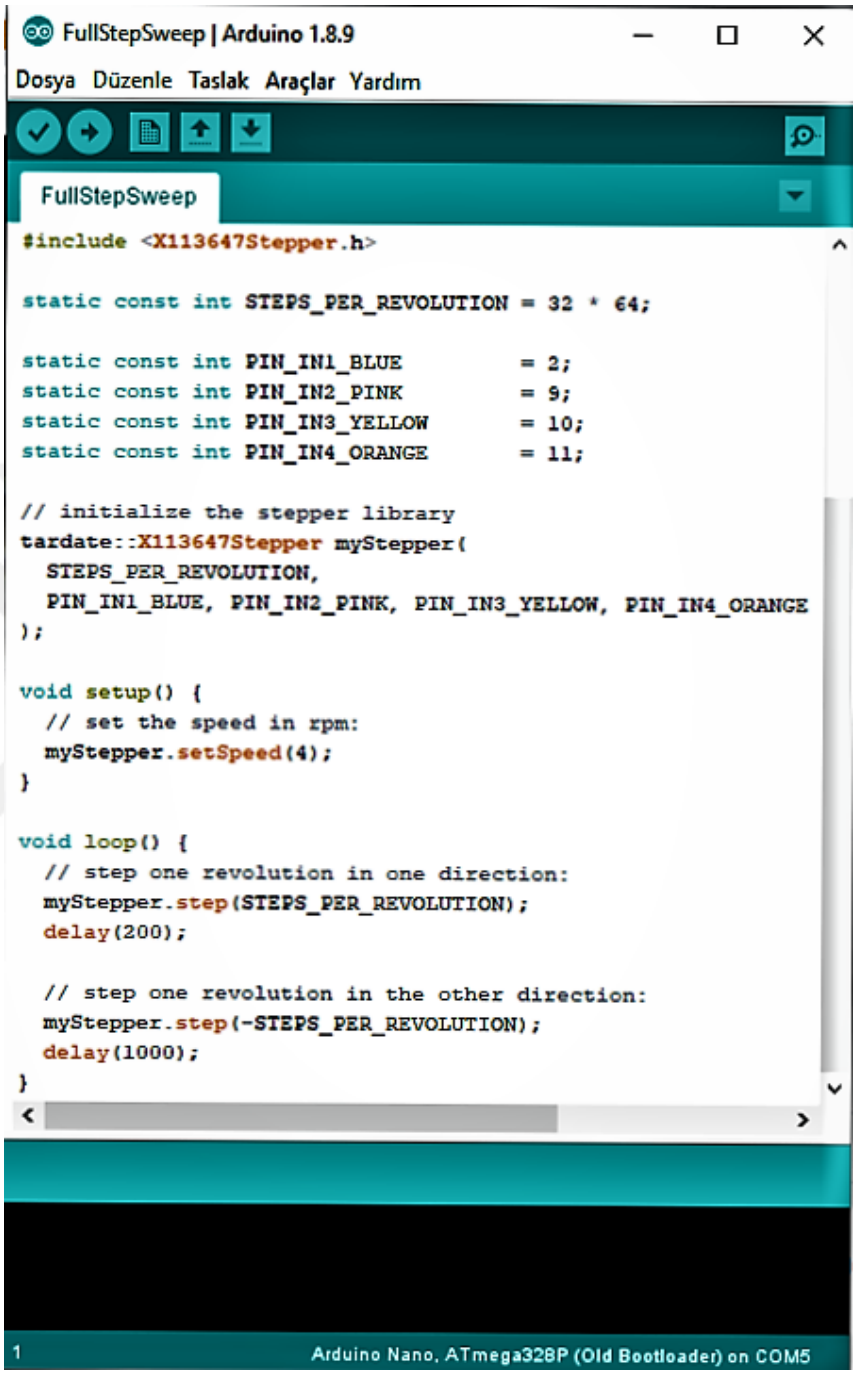
Voulodimos, A., Doulamis, N., Doulamis A. ve Protopapadakis, E. (2018). Deep Learning for Computer Vision: A Brief Review. *Computational Intelligence and Neuroscience.* 2018: 1-13.

Wang, Z. (2015). The Applications Of Deep Learning On Traffic Identification. https://www.blackhat.com/docs/us-15/materials/us-15-Wang-The-Applications-Of-Deep-Learning-On-Traffic-Identification.pdf, (26.12.2019).

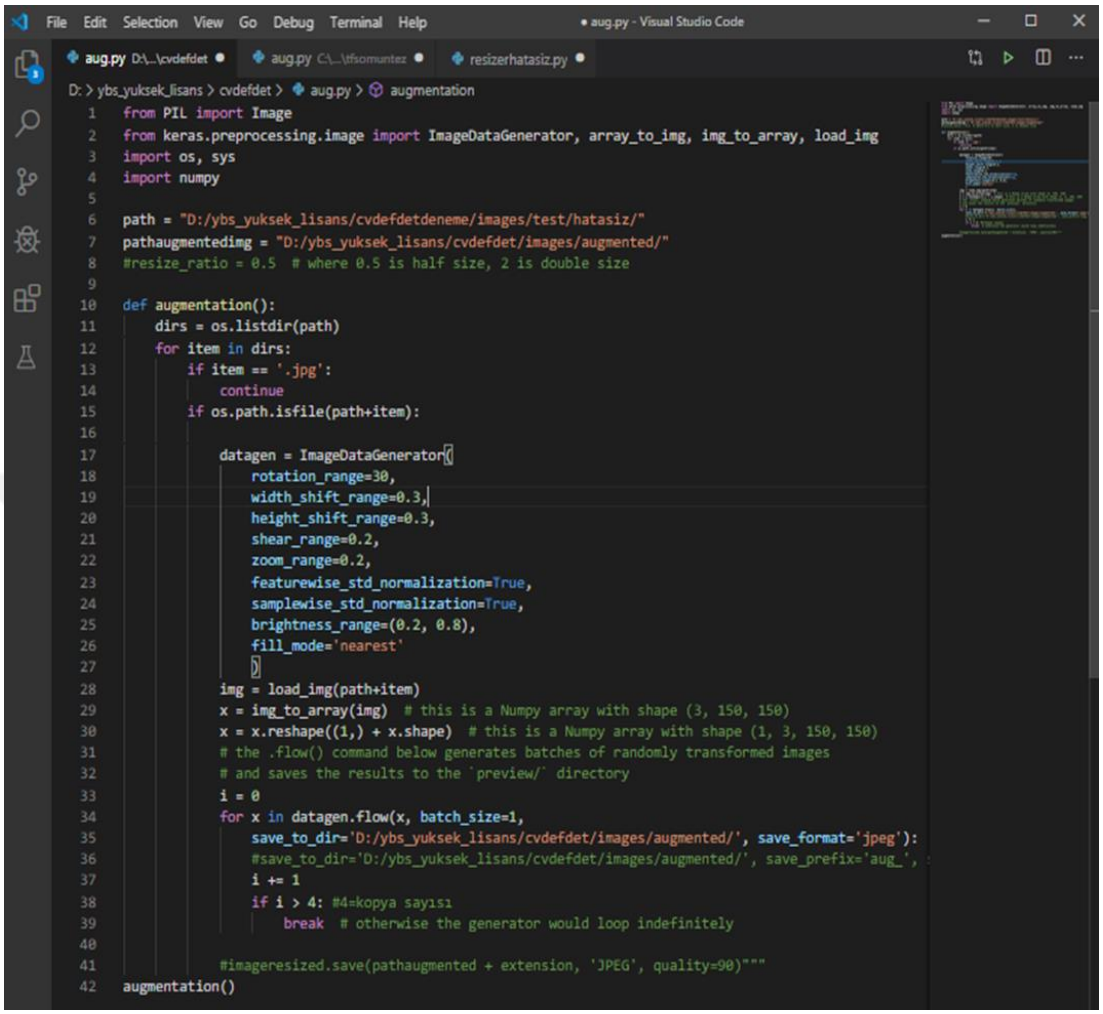
Zocca, V., Spacagna, G., Slater, D., Roelants, P. (2017). *Python Deep Learning*. Birleşik Krallık: Packt Publishing.

EKLER

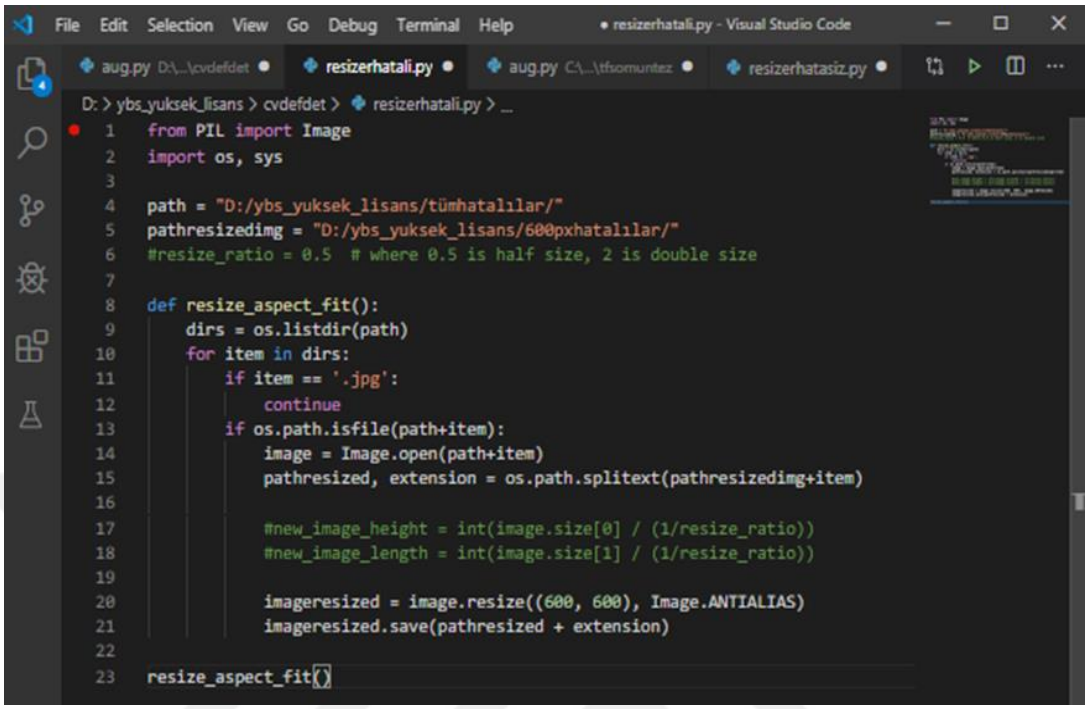
Ek 1: Step Motor Kodları



Ek 2: Veri Artırım Kodları



Ek 3: Yeniden Boyutlandırma Kodları



Ek 4: Google Colaboratory Kodları

```
# -*- coding: utf-8 -*-
"""traincoco.ipynb

Automatically generated by Colaboratory.

Original file is located at
    https://colab.research.google.com/drive/1U4S2qvImJGUImY5fAGR6wEg1laaH3Hy8
"""
# Load the Drive helper and mount
from google.colab import drive
# This will Prompt for authorization.
drive.mount('/content/drive')

!ls "/content/drive/My Drive/tensorflow2"

!pip uninstall tensorflow

#!pip3 install grpcio==1.25.0
!pip3 install tensorflow==1.13.2
!pip3 install tensorflow-gpu==1.13.2
!pip3 install tensorboard
!apt-get install protobuf-compiler python-pil python-lxml python-tk
!pip3 install Cython
!pip3 install contextlib2
!pip3 install jupyter
!pip3 install matplotlib
!pip3 install pandas
!pip3 install opencv-python
!pip3 install tensorboardcolab==1.13.2

!pip3 list | grep tensorflow

!git clone https://github.com/cocodataset/cocoapi.git
!cd cocoapi/PythonAPI; make; cp -r pycocotools /content/drive/My\ Drive/tensorflow2/models/research

cd /content/drive/My Drive/tensorflow2/models/research

!mkdir train eval

# Commented out IPython magic to ensure Python compatibility.
# %set_env PYTHONPATH=/content/drive/My Drive/tensorflow2/models:/content/drive/My Drive/tensorflow2/models/
research:/content/drive/My Drive/tensorflow2/models/research/slim

!protoc object_detection/protos/*.proto --python_out=.

!python setup.py build

!python setup.py install

!python object_detection/builders/model_builder_test.py

cd object_detection

!python xml_to_csv.py

!python generate_tfrecord.py --csv_input=images/train_labels.csv --image_dir=images/train --
output_path=train.record

!python generate_tfrecord.py --csv_input=images/test_labels.csv --image_dir=images/test --
output_path=test.record

cd /content/drive/My Drive/tensorflow2/models/research/object_detection

!pip install -U tensorboardcolab

!pip install npm

from tensorboardcolab import *
```

```
tensorboard --logdir=training/ --host=0.0.0.0

!python train.py --logtostderr --train_dir=training/ --
pipeline_config_path=training/faster_rcnn_inception_v2_pets.config

!python export_inference_graph.py --input_type image_tensor --
pipeline_config_path training/faster_rcnn_inception_v2_pets.config --
trained_checkpoint_prefix training/model.ckpt-28327 --output_directory inference_graph

"""kaynak : https://github.com/tensorflow/models/blob/master/research/object_detection/object_detection_tuto
rial.ipynb

!python Object_detection_image.py #Test resmi üzerinden detection işlemi.
!python Object_detection_webcam.py #Webcam ile alınan gerçek zamanlı görüntü üzerinden detection işlemi.
```

Ek 5: Uygulama Geliştirme Esnasında Yaşanan Hatalar

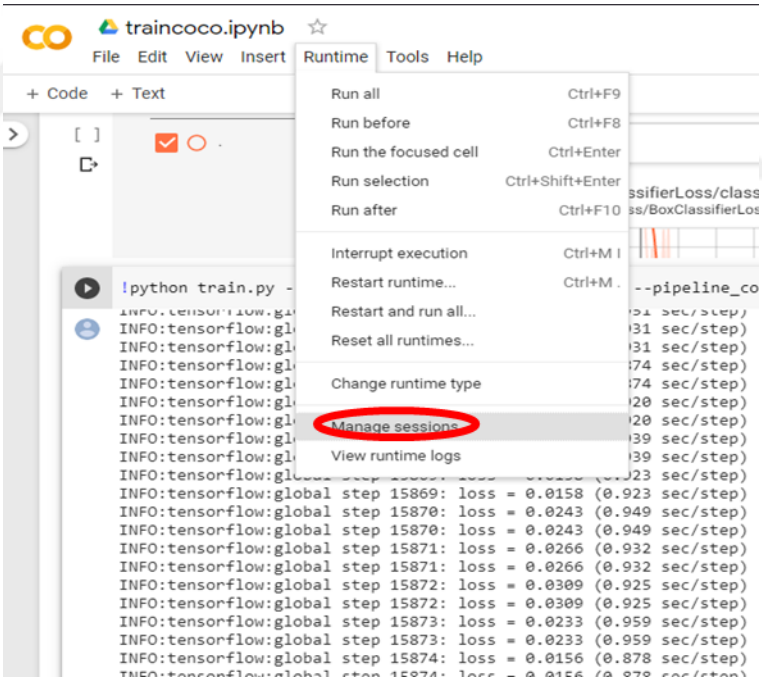
Hata 1: Tensorboard, Google Colab üzerinde çalıştırıldığında hata alınıyordu ve pencere açılmıyordu.

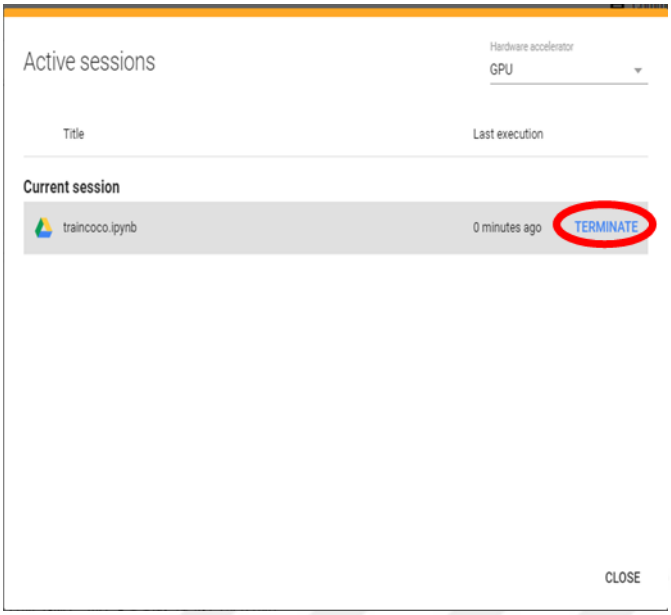
Aşağıdaki kod ile host adresi 0.0.0.0. olarak düzenlendikten sonra tensorboard googlecolab üzerinden bizim server url'mize ve portumuzla bağlanmış ve sorunsuz çalışmaya başlamıştır.

tensorboard --host 0.0.0.0 <other args here>

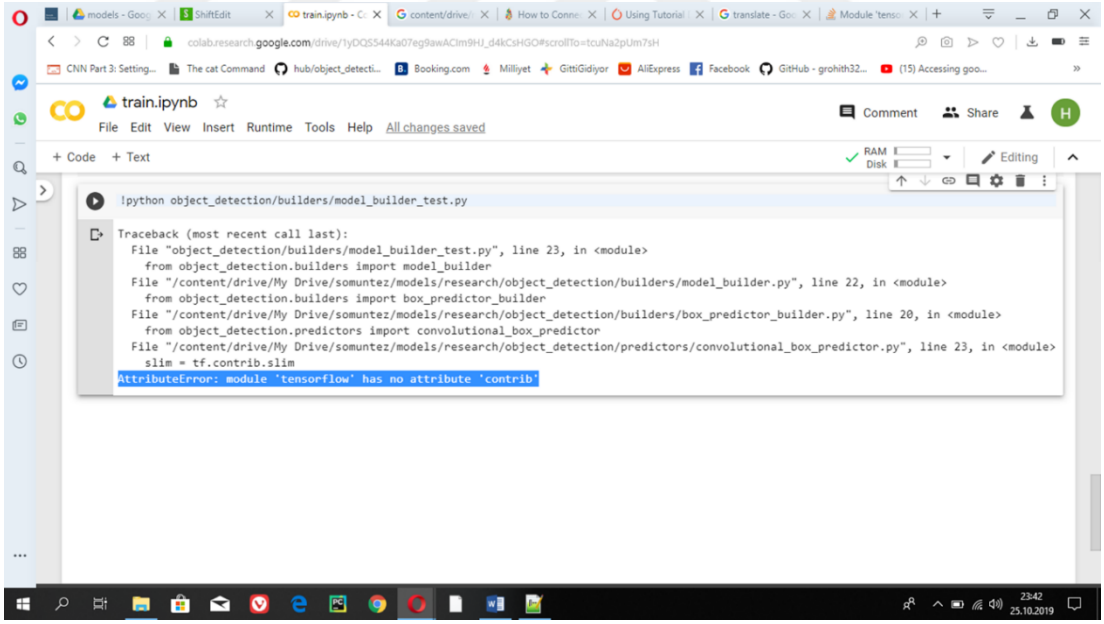
--host 0.0.0.0 host'u tensorflow'a, yerel makinedeki tüm Ipv4 adresleri üzerindeki bağlantıları dinlemesini söyler.

Hata 2: Tensorboard'un, runtime süresi 12 saat geçince donduğu ve sayfadan attığı görülmüş olup, böyle bir durumla karşılaşıldığında aşağıdaki yöntemle kolayca çözüldüğü tespit edilmiştir.





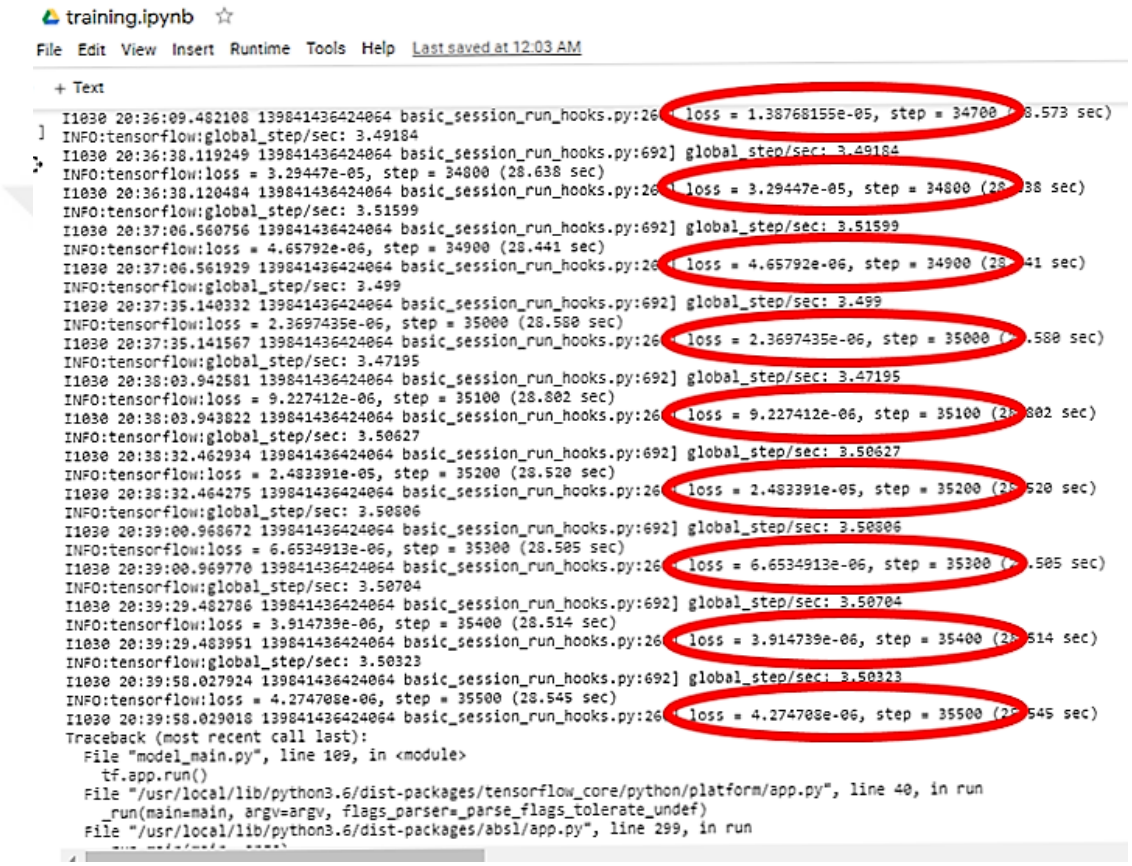
Hata 3: Tensorflow 2.0 kurulundan sonra, model_builder_test.py python modülü çalıştırıldığında aşağıdaki hata alınmıştır.



Tensorflow 2.0 ile Tensorflow1.x versiyonlarında kullanılan Contrib modülü kullanımına son verildiği tespit edilmiştir (https://stackoverflow.com/questions/55870127/module-tensorflow-has-no-attribute-

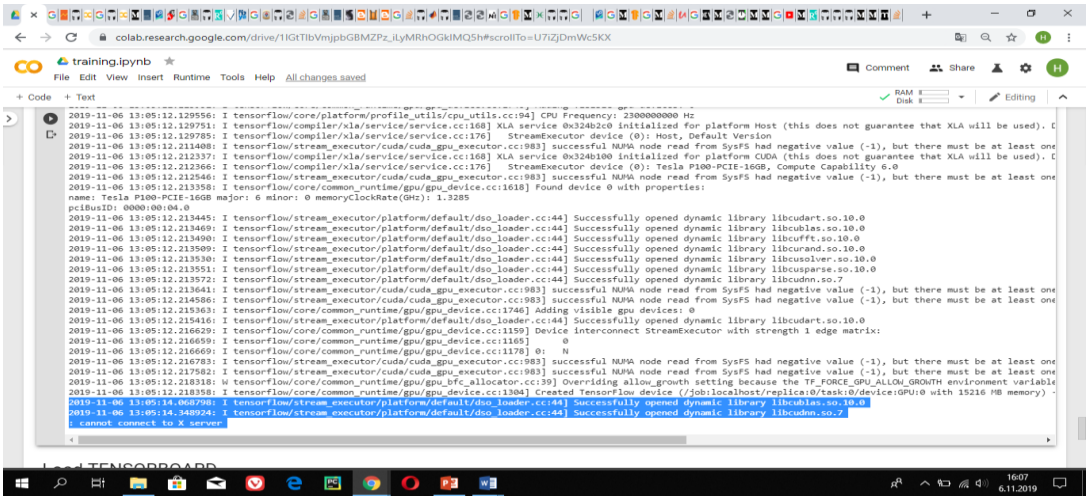
contrib). Tensorflow 2.0 kurulumu kaldırılmış ve yerine Tensorflow1.13.2 versiyonu kurularak bu hata çözülmüştür.

Hata 4: Eğtim esnasında loss değerlerinde büyük sapmalar meydana gelmiştir. 3000'inci adımdan sonra yavaş yavaş loss değeri 4-6-7-9 görülmeye başlanmış ve adım sayısı arttıkça bu değer 9'lara kadar sık sık çıkmıştır. 35500 adım eğitilmiş ama loss değeri 1'in altına hiç düşmemiştir.



Bu sapmaların nedeninin ezberleme (overfitting) mi olduğu yoksa kodlarla ilgili konfigürasyonunda mı sorun olduğu araştırılmıştır. Bu noktada sorunun, sınıf sayısının 2 yazılarak yalnızca 1 etiket belirtilmesinden kaynaklandığı düşünülmüştür. Ardından hatasiz fotoların da etiketlenmesi, label_img.pbtx ve config dosyalarının 2 sınıf 2 etiket olacak şekilde yeniden düzenlenerek eğitilmesiyle sorun çözülmüştür.

Hata 5: Object_detection_image.py modülü çalıştırıldığında : cannot connect to X server hatası alınması.



Eğitim işleminin tensorflow 1.13.2 versiyonunda yapılmasına rağmen, nesne tespit modülünün tensorflow 2.0 versiyonu üzerinde çalıştırıldığı anlaşılmıştır. Nesne tespit modülü tensorflow 1.13.2 üzerinde tekrar çalıştırılınca hata alınmamıştır.

Hata 6: ModuleNotFoundError: No module named 'deployment' hatası.

Bu hatanın sebebinin PYTHONPATH'in doğru yazılmadığında ortaya çıktığı görülmüştür.

Hata 7: TypeError: __init__() got an unexpected keyword argument 'dct_method'

Bu hatayı düzeltmek için /object_detection/data_decoders/tf_example_decoder.py dosyası açılarak 110'uncu satırda "dct_method=dct_method" argümanı silinmiştir.

Hata 8: ImportError: cannot import name 'preprocessor_pb2' (ve pb2 ile ilgili diğer hatalar)

Bu hatanın derlenmeyen .proto dosyası olduğunda meydana geldiği anlaşılmıştır. /object_detection/protos klasörü kontrol edilmiş ve eksik olan proto dosyasının protobufları derlediğimiz komuta eklenmiş ve kod tekrar çalıştırılmıştır.

Hata 9: Unsuccessful TensorSliceReader constructor: Failed to get "file path" The filename, directory name, or volume label syntax is incorrect.

Bu hatanın, config dosyasının yanlış düzenlediğinde alındığı görüşmüştür. Doya uzantıları düzeltilerek ve düz slash(/) kullanılarak sorun çözülmüştür.

Hata 10: ModuleNotFoundError: No module named 'object_detection.legacy'Tensorflow Object Detection API'ın en son güncellenmiş versiyonu indirilmiş ve bu versiyonda legacy klasörü yüklenmiştir.

Hata 11: TypeError: a bytes-like object is required, not 'str' labelmap.pbtxt UTF-8'e çevrilerek düzeltilmiştir.