KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

LİSANS TEZİ

EL ÇİZİMİ AKIŞ DİYAGRAMLARININ DİJİTALE AKTARIMI

SERKAN ÖZDEMİR, AZİZ BATUHAN BIYIKLI, AZİZ YELBAY

KOCAELİ 2021

KOCAELİ ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ

BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

BİTİRME ÇALIŞMASI

EL ÇİZİMİ AKIŞ DİYAGRAMLARININ DİJİTALE AKTARIMI

SERKAN ÖZDEMİR, AZİZ BATUHAN BIYIKLI, AZİZ YELBAY

Doç. Dr. Ahmet Sayar
Danışman, Kocaeli Üniv. .....
Dr. Öğr. Üyesi. Pınar Onay Durdu
Jüri Üyesi, Kocaeli Üniv. .....
Ar. Gör. Seda Kul
Jüri Üyesi, Kocaeli Üniv. .....

Tezin Savunulduğu Tarih: 12.06.2021

ÖNSÖZ VE TEŞEKKÜR

Bu tez çalışması, El Çizimi Akış Diyagramlarının Dijitale Aktarımı amacıyla gerçekleştirilmiştir.

Tez çalışmamızda desteğini esirgemeyen, çalışmalarımıza yön veren, bize güvenen ve yüreklendiren danışmanımız Doç. Dr. Ahmet Sayar'a sonsuz teşekkürlerimizi sunarız.

Hayatımız boyunca bize güç veren en büyük destekçilerimiz, her aşamada sıkıntılarımızı ve mutluluklarımızı paylaşan sevgili ailelerimize teşekkürlerimizi sunarız.

Mayıs – 2021

Serkan ÖZDEMİR
Aziz Batuhan BIYIKLI
Aziz YELBAY

ETİK BEYAN

Bu tezdeki bütün bilgileri etik davranış ve akademik kurallar çerçevesinde elde ettiğimi ve tez yazım kurallarına uygun olarak hazırlanan bu çalışmada bana ait olmayan her türlü ifade ve bilginin kaynağına eksiksiz atıf yaptığımı bildiririm. İfade ettiklerimin aksi ortaya çıktığında ise her türlü yasal sonucu kabul ettiğimi beyan ederim.

Öğrenci No: 170201035
Adı Soyadı: Serkan Özdemir
İmza: .....

Öğrenci No: 170201090
Adı Soyadı: Aziz Batuhan Bıyıklı
İmza: .....

Öğrenci No: 170201046
Adı Soyadı: Aziz Yelbay
İmza: .....

İÇİNDEKİLER

ŞEKİLLER DİZİNİ

TABLOLAR DİZİNİ

SİMGELER VE KISALTMALAR DİZİNİ

y : Çıkış vektörü
pc : Çerçeve içinde nesnenin olma ihtimali
bx : Tespit edilen nesnenin merkez noktasının x bileşeni
by : Tespit edilen nesnenin merkez noktasının y bileşeni
bw : Tespit edilen nesnenin genişliği
bh : Tespit edilen nesnenin yüksekliği
c : Tespit edilen nesnenin ait olduğu sınıf numarası

Kısaltmalar

CNN :Convolutional Neural Network (Evrişimli Sinir Ağları)
CUDA :Compute Unified Device Architecture
CUDNN :CUDA Deep Neural Network Library
GPU :Graphics Processing Unit (Grafik İşlemci Birimi)
IoU :Intersection over Union (Birleşim üzerinde Kesişim)
YOLO :You Only Look Once(Sadece Bir Kez Bak)
R-CNN :Region Based CNN (Bölge Tabanlı CNN)
XML :Extensible Markup Language (Genişletilebilir İşaretleme Dili)

ÖZET

İş dünyasında çeşitli alanlardaki işlem ve uygulamaların yönetilmesi, belgelendirilmesi, tasarlanması ve çözümlenmesinde akış şemaları sıklıkla kullanılmaktadır. Bu şemalar işletmelerdeki süreçlerin düzenlenmesine ve verimliliğin artmasına katkı sağlamaktadır.

Bu şemalar elektronik ortamlarda çizimi kolaylaştıran uygulamalar yardımıyla çizilebilsede gerek toplantılarda gerekse taslak oluşturma sırasında el ile de çizilmektedir. El ile taslak halinde çizilmiş bir akış şeması geliştirilmek veya saklanmak istediğinde djital ortama aktarma problemi ortaya çıkmaktadır. Bu proje el ile çizilen bir akış diyagramının, üzerinde düzeltmeler yapılabilecek uygun bir platform için dijitale dönüştürülmesini amaçlanmıştır. Akış şemalarının büyük bir çoğunluğunu oluşturan temel şekillerin makine öğrenimi ile tanınması ve tanınan nesnelerin XML formatına dönüştürülerek dijital platforma aktarılması gerçekleştirilmiştir.

Anahtar Kelimeler: Makine öğrenimi , YOLO , Nesne Tespiti, Akış Diyagramı , Dijitalleştirme

ABSTRACT

Flow charts are frequently used in the management, documentation, design and analysis of processes and applications in various fields in the business world. These schemes contribute to the regulation of processes in enterprises and to increase efficiency. Although these diagrams can be drawn with the help of applications that facilitate drawing in electronic media, they are also drawn by hand both in meetings and during draft creation. When a flowchart drawn in hand-drawn form wants to be developed or stored, the problem of transferring it to digital media arises. This project aims to transform a flowchart drawn by hand into digital for a suitable platform on which corrections can be made. Recognition of the basic shapes that make up the majority of the flow charts with machine learning and the transfer of the recognized objects to the digital platform by converting them into XML format.

Keywords: Machine learning, YOLO, Object Detection, Flowchart, Digitization

1.GİRİŞ

Akış şemaları iş süreçlerinin yönetilmesi ve verimliliğin artması amacıyla kullanılmaktadır. Bir akış şemasını, bir işlemi, sistemi, süreci veya bir algoritmayı gösteren şekiller bütünü olarak tanımlanabilir. Karmaşık bir süreci basitleştiren ve anlaşılmasını kolay hale getiren bu diyagramlar çoğu zaman çizimi kolaylaştıran dijital ortamlarda düzenlenir. Ancak bu diyagramların çizimi kağıt üzerinde taslaklar şeklinde başlar ve bu taslakların dijitale aktarılması gerekir. Kağıt üzerine el ile çizilmiş bir akış şemasının, dijital çizim programlarına uygun bir şekilde dijitale aktarılması ihtiyacı için bu tez içerisinde açıklanacak olan projenin geliştirilmesine karar verilmiştir. Bu proje akış diyagramlarının büyük çoğunluğunun tanınmasını sağlayacak temel 7 şekil [1] baz alınarak geliştirilmiştir.

Akış şemalarının dijitalleştirilmesi iki adımdan oluşur şemada yer alan şekillerin tanınması ve bu şekillerin dijitale aktarılması. Bu alanda yapılan benzer çalışmalar şu şekildedir:

-Bernhard Schafer ve Heiner Stuckenschmidt tarafından geliştirilen ve çevrim içi çizimlerde kalem vuruşlarının oluşturduğu şekillere, çevrim dışı tanımlamada ise CNN algoritması tarafından şekillerin tanınmasına dayalı “Arrow R-CNN” projesi [2].

-Changcheng Xiao, Changhu Wang, Liqing Zhang tarafından geliştirilen ve akıllı telefonlar tarafından çekilen akış diyagramlarının anlık olarak tanınmasını sağlayan ve powerpoint dosyası ile çıktı oluşturan “PPTLens” sistemi[3].

-Jie Wu, Changhu Wang, Liqing Zhang, Yong Rui’nin çevrim dışı çalışan ve görüntüdeki nesnelerin konturlarını algılayarak hangi nesne olduğuna karar veren bir algoritma geliştirdikleri “Offline Sketch Parsing via Shapeness Estimation” isimli çalışma[4].

-Plimmer ve Freeman tarafından geliştirilen ve kullanıcıların dijital ortamda çizdikleri akış diyagramını koda dönüştürmeyi hedefleyen InkKit projesi[5].

-S. Chakraborty, S. Paul, ve Sk. Md. Masudul Ahsan’ın beyaz zemin üzerine çizilmiş akış diyagramlarını morfolojik işlemler ile belirgin bir hale getirip nesne tanıma algoritması ile şekillerin ve bu şekillere ait koordinatların XML koda dönüştürülmesini sağladıkları proje[6].

Bu projeler ile geliştirdiğimiz proje arasındaki farklar :

-Dijital ortamda veya kağıt üzerinde olması fark etmeksizin el ile çizilmiş diyagram görüntüsünde nesne tanıma algoritması kullanılarak diyagramın dijitalleştirilmesi.

-Benzer çalışmalarda çevrim dışı durumda görülen ve nesne tanıma için kullanılan CNN algoritması yerine daha gelişmiş, hızlı ve yüksek doğruluk oranına sahip YOLO algoritmasının kullanılması.

Benzer bir kullanım olarak S.Chakraborty ve diğerlerinin de kullandığı ve dijital çıktı olarak düzeltilmiş çıktı görüntüsü yerine dijital platformda düzenlenebilir bir XML çıktısının üretilmesi.

Bu tezde akış şemalarında yer alan nesnelerin tespiti için YOLO algoritması kullanılmıştır. Gerekli veri seti proje paydaşları tarafından çizilip algoritma için etiketlenmiştir. Eğitim süreci Google Colab hizmeti ile başlamış ancak çeşitli sebeplerden dolayı bilgisayar ortamında devam edilmiştir. Eğitilen model ile el çizimi akış şeması görseli üzerindeki nesneler ayırt edilmiş ve bu nesnelere ait koordinat bilgileri elde edilmiştir. Bu bilgiler çizim platformu olarak baz aldığımız draw.io için xml formatına dönüştürülmüş ve dijitalleştirme işlemi tamamlanmıştır.

Bu tez şu şekilde düzenlenmiştir ; Yararlanılan teknojiler 2. kısımda incelenmiştir. Veri setinin oluşturulması, modelin eğitilmesi, test,geliştirme ve XML koda dönüştürülmesi süreçleri 3. kısımda açıklanmıştır. 4. kısımda ise elde edilen sonuçlara yer verilmiştir.

2.KULLANILAN TEKNOLOJİLER

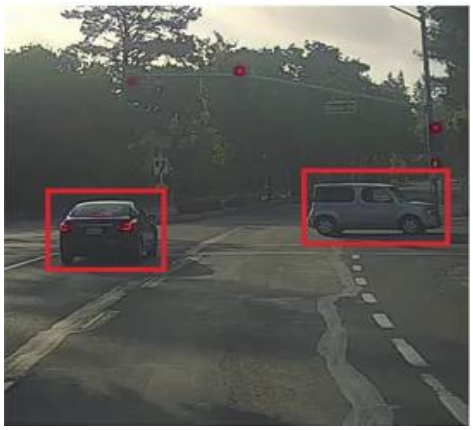
Akış diyagramının dijitalleştirilmesi amacı taşıyan projenin nesneleri tespit etme kısmında YOLO algoritması kullanılmıştır.

2.1.YOLO Algoritması

YOLO'dan önce CNN teknik ailesi [7] , görüntü içindeki nesneleri yerelleştirmek için bölgeleri kullanır. Görüntünün tamamına bakmak yerine yalnızca görüntülerin bir nesneyi içerme şansı daha yüksek olan kısımlarına bakar. YOLO(You Only Look Once) [8,9,10] çerçevesi, nesne algılamayı farklı bir şekilde ele alır. Tüm görüntüyü tek bir örnekte alır ve bu kutular için sınırlayıcı kutu koordinatlarını ve sınıf olasılıklarını hesaplar. YOLO kullanmanın en büyük avantajı yüksek hızıdır. Ayrıca YOLO hızlı olmasına rağmen nesne tespitinde de oldukça başarılıdır.

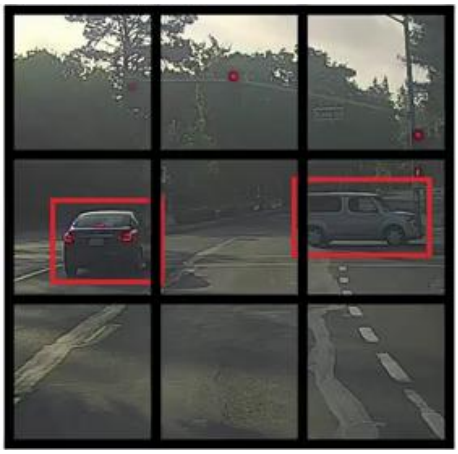
YOLO'nun çalışma mantığı adım adım şu şekilde işlemektedir [11] ;

YOLO önce bir girdi görüntüsü alır



Şekil 2.1 Girdi Görüntüsü [11]

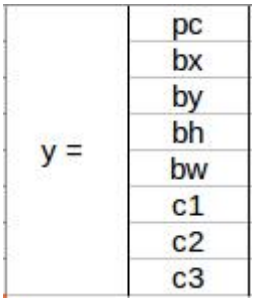
- Algoritma giriş görüntüsünü ızgaralara böler 3x3 ızgara için örnek görüntü şu şekilde oluşur :



Şekil 2.2 Izgaralara Ayrılmış Görüntü [11]

- Her ızgarada görüntü sınıflandırma ve yerelleştirme (koordinat bulma) uygulanır. YOLO ızgara içerisinde nesne varsa bu nesneye ait sınıf tahmini ve sınırlayıcı kutuları tahmin eder.

Görüntünün 3 X 3 boyutunda bir ızgaraya bölündüğünü ve nesnelerin sınıflandırılması istenen toplam 3 sınıf olduğu (yaya, araba, motosiklet) varsayıldığında. Dolayısıyla, her bir ızgara hücresi için y çıktı etiketi sekiz boyutlu bir vektör olacaktır:



Şekil 2.3 Çıktı Etiketi [11]

Burada,

- pc, ızgarada bir nesnenin olup olmadığını tanımlar (olasılıktır)
- bx, by, bh, bw bir nesne varsa sınırlayıcı kutuyu belirtir
- c1, c2, c3 sınıfları temsil eder. Yani, belirtilen örneğe göre nesne bir araba ise, c2 1 olacak , c1 ve c3 0 olacaktır.Bu tüm ızgara parçaları için uygulanacaktır.

YOLO'nun ızgarada gerçekten bir nesne olup olmadığına nasıl karar verdiğini anlamak önemlidir. Etiketli eğitim verisine göre YOLO nesnenin orta noktasını nesnenin varlığını belirleyici unsur olarak kabul eder. Ele alınan örnekte iki nesne (iki araba) vardır, bu nedenle YOLO bu iki nesnenin orta noktasını alacak ve bu nesneler bu nesnelerin orta noktasını içeren ızgaraya atanacaktır. Mevcut örneğe göre sol üst ve sol orta hücreleri incelenirse:

Şekil 2.4 Örnek Izgara Bölmeleri [11]

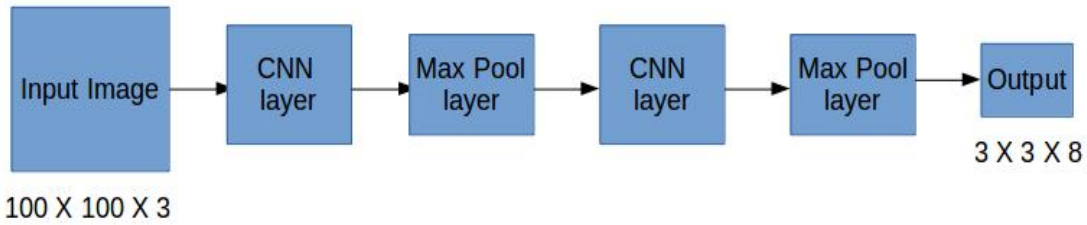
Birinci hücrede nesne olmadığından, pc=0 ikinci hücrede ise pc=1 olacaktır y görünümleri ise şu şekilde olacaktır:

y =	0	y =	1
	?		bx
	?		by
	?		bh
	?		bw
	?		0
	?		1
	?		0

Şekil 2.5 Örnek Bölmeler İçin Çıkış Etiketleri [11]

Burada "?" hücre içerisinde nesne olmamasından kaynaklanmaktadır. bx, by, bh, bw, c1, c2 ve c3'ün ne içerdiğinin önemli olmadığı anlamına gelir. İçinde araba olan hücrede ise bir nesne olduğu için pc 1'e eşit olacaktır. Bx, by, bh, bw ilgilendiğimiz belirli ızgara hücresine göre hesaplanacaktır. Araba ikinci sınıf olduğundan, c2 = 1 ve c1 ve c3 = 0.

Sonuç olarak , 9 ızgaranın her biri için sekiz boyutlu bir çıktı vektörü olacaktır. Bu çıktının şekli 3 X 3 X 8 olacaktır. Şimdi bir giriş resmimiz var ve ona karşılık gelen hedef vektör. Yukarıdaki örneği kullanarak (giriş resmi - 100 X 100 X 3, çıktı - 3 X 3 X 8), modelimiz aşağıdaki gibi eğitilecektir:



Şekil 2.6 Model Görünümü [11]

Modelin eğitimi için hem ileri hem de geri yayılım yapılır. Yani başlangıç olarak bir ağırlık değeri alınır. İlk ileri yayılımın ardından ağırlıklar güncellenir (geri yayılım). Test aşamasında, modele bir görüntü iletilir ve bir y çıktısı elde edene kadar ileriye yayılım olur.

Bir nesne birden fazla ızgaraya yayılsa bile, yalnızca orta noktasının bulunduğu tek bir ızgaraya atanacaktır. Izgara büyüklüğü arttırılarak (örneğin 19 X 19) aynı ızgara hücresinde birden fazla nesnenin görünme şansı azaltılabilir.

Nesneye ait sınırlayıcı kutuların belirlenmesi için kullanılan bx, by, bh ve bw değerleri ilgilenilen ızgara hücresine göre hesaplanır. Bu kavramı örnek üzerinde bir araba içeren orta-sağ hücre için ele alırsak:

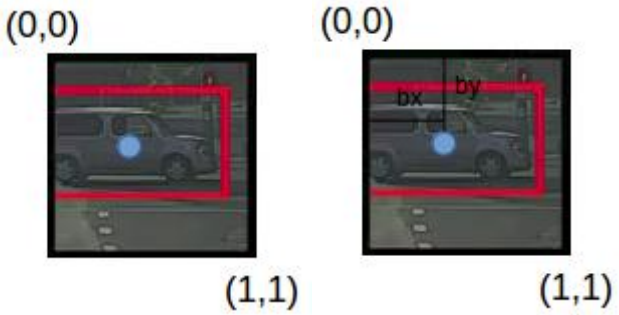
Şekil 2.7 Nesnenin Çerçeve İçine Alınmış Görüntüsü [11]

bx, by, bh ve bw yalnızca bu ızgaraya göre hesaplanacaktır. Bu ızgara için y etiketi şöyle olacaktır:

y =	1
	bx
	by
	bh
	bw
	0
	1
	0

Şekil 2.8 y Çıktı Etiketi [11]

pc = 1 çünkü bu ızgarada bir nesne var ve bu bir araba olduğu için, c2 = 1. Bu durumda, bx, by, bh ve bw'ye ait değerlerin hesaplanması için hücreye ait koordinatlar kullanılır. YOLO'da, tüm ızgaralara atanan koordinatlar şu şekildedir:



Şekil 2.9 Çizilen Çerçevenin Koordinat ve Orta Nokta Gösterimi [11]

bx, by bu ızgaraya göre nesnenin orta noktasının x ve y koordinatlarıdır. Bu durumda, (yaklaşık) bx = 0.4 ve = 0.3 olacaktır. bh, sınırlayıcı kutunun yüksekliğinin (yukarıdaki örnekte kırmızı kutu) karşılık gelen ızgara hücresinin yüksekliğine oranıdır, bu durumda yaklaşık 0,9'dur. Yani, bh = 0.9. bw, sınırlayıcı kutunun genişliğinin ızgara hücresinin genişliğine oranıdır. Yani, bw = 0.5 (yaklaşık olarak). Bu ızgara için y etiketi şöyle olacaktır:

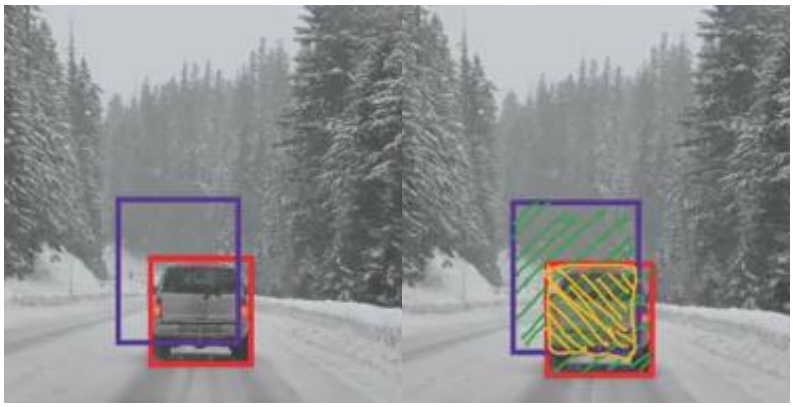
y =	1
	0.4
	0.3
	0.9
	0.5
	0
	1
	0

Şekil 2.10 Hesaplanmış Değerler ile Çıktı Etiketi [11]

Burada, orta nokta her zaman ızgara içinde kalacağından, bx ve by'nin her zaman 0 ile 1 arasında değişecektir. Oysa bh ve bw, sınırlayıcı kutunun boyutlarının ızgaranın boyutundan fazla olması durumunda 1'den fazla olabilir.

Birleşim Üzerinden Kesişim ve Maksimum Olmayanı Bastırma Yöntemleri

Tahmin edilen çerçevenin ne kadar başarılı olduğunun hesaplanması için birleşim üzerinden kesişim tekniği kullanılır. Bu teknik gerçek çerçeve ile tahmin edilen çerçevenin birleşimi üzerinden kesişimini hesaplar. Bir araba için gerçek ve tahmin edilen çerçeveleri aşağıda gösterildiği gibi düşünübiliriz:



Şekil 2.11 Gerçek ve Tahmin Edilen Çerçeveler ve Bu Çerçevelerin Kesişimleri [11]

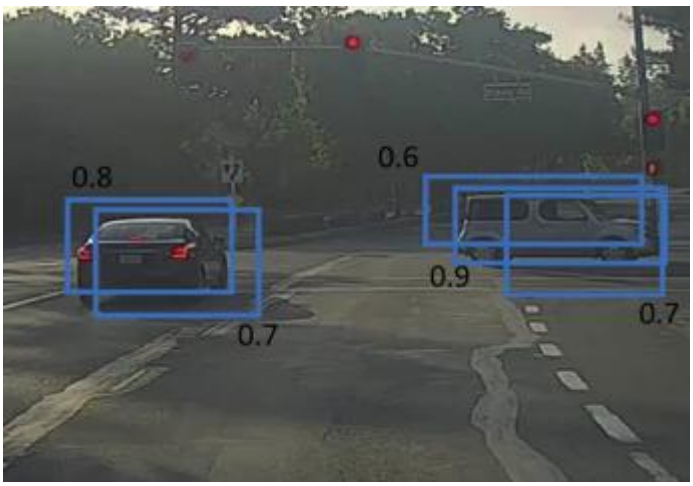
Burada, kırmızı kutu gerçek sınırlayıcı kutudur ve mavi kutu öngörülendir. İyi bir tahmin olup olmadığına karar vermek için IoU (Intersection over Union) değeri hesaplanır , bu iki kutunun birleşimi üzerinden kesişme alanının hesaplanmasıdır.. Bu alan:

IoU = kesişim / birleşim yani ,

IoU = Sarı kutu alanı / Yeşil kutların alanı

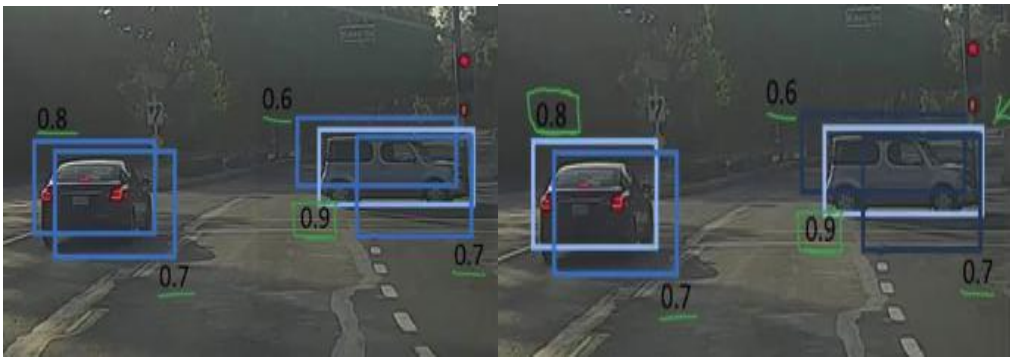
IoU 0,5'ten büyükse, tahminin yeterince iyi olduğunu söyleyebiliriz. 0.5 burada aldınan rasgele bir eşik değeridir , probleme göre değiştirilebilir. Sezgisel olarak, eşik değer ne kadar yükseltilirse, tahminler o kadar gerçeğe yakın olur. YOLO'nun çıktısını iyileştirmek için önemli bir teknik daha vardır.Maksimum Olmayan Bastırma [12].

Nesne algılama algoritmalarındaki en yaygın sorunlardan biri, bir nesneyi yalnızca bir kez algılamak yerine, birden çok kez algılayabilmeleridir.



Şekil 2.12 IoU Değerleri ile Çizilen Çerçeveler [11]

Burada arabalar için birden fazla çerçeve çizildiğini görünmektedir. Maksimum Olmayan Bastırma tekniği bunu temizler, böylece nesne başına yalnızca tek bir algılama elde ederiz. Bu yaklaşım önce her tespit ile ilişkili olasılıklara bakar ve en büyüğünü alır. Yukarıdaki resimde, 0,9 en yüksek olasılıktır, bu nedenle önce 0,9 olasılıklı kutu seçilecektir:



Şekil 2.13 IoU Değeri Yüksek Olan Çerçevelerin Öne Çıkarılması [11]

Önce 0.9 olasıklı kutu ile çakışan diğer kutular bastırılır. Ardından görsel üzerindeki diğer yüksek IoU değerli kutuya geçilir. Bu görselde 0.8 değerli kutudur. Aynı işlem bu kutu için de yapılır ve çakışan çerçeveler bastırılır.

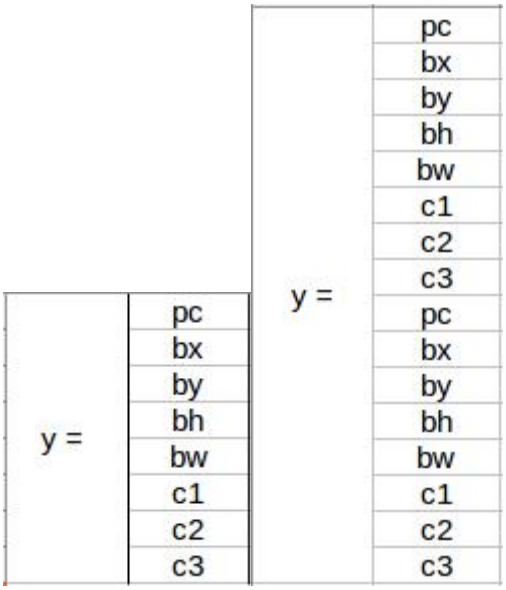
Şekil 2.14 Maksimum IoU Değerine Sahip Çerçeveler [11]

Sonuç olarak her bir nesne için en yüksek orana sahip çerçeve kalır. Maksimum olmayanı bastırma metodu bu amaçla kullanılır. YOLO algoritmasında kullanılan bir diğer metod ise bağlantı kutularıdır (Anchor Boxes).

Görselin kutulara ayrılmasının ardından her bir parçanın, içerisinde tek bir nesneye ait merkez nokta olması ve y değerinin bu nesneye ait değerleri tuması nedeniyle yalnızca bir nesneyi tanımlayabildiği bir durum söz konusudur. Ancak bir kutu içerisinde birden fazla nesnenin merkez noktası olursa tüm nesnelerin tespiti için bir çözüm yolu gereklidir. BU çözüm yolu YOLO içerisinde bağlantı kutuları (anchor boxes) olarak ifade edilir.

Şekil 2.15 Çakışan İki Nesnenin Bağlantı Kutuları ile Gösterimi [11]

Yukarıdaki örnekte, her iki nesnenin orta noktası aynı ızgarada yer almaktadır. Normal şartlarda bu nesnelerden yalnızca biri alınabilir. Ancak bağlantı kutuları kullanılırsa, her iki kutunun da çıktısı alınabilir.



Şekil 2.16 Bağlantı Kutuları Kullanıldığında y Çıktı Etiketi [11]

İlk 8 sıra bağlantı kutusu 1'e aittir ve kalan 8 sıra bağlantı kutusu 2'ye aittir. Nesneler bağlantı kutularına sınırlayıcı kutuların benzerliğine ve bağlantı kutusu şekline göre atanır. Bu durumda çıktı, 3 X 3 X 8 yerine (3 X 3 ızgara ve 3 sınıf kullanarak), 3 X 3 X 16 olacaktır (çünkü bağlantılı iki kutu kullanılmakta). Böylece, her bir ızgara için, bağlantılı kutu sayısına bağlı olarak iki veya daha fazla nesne tespit edilebilir.

2.2.Python ElementTree

Python standart kütüphanelerinden birisidir. Kullanımı oldukça kolaydır ve kullanım alanı yaygındır. XML dosyasını list yapısında, elemanları dict yapısında sunar. Özet bir tabirle daha Pythonic bir kullanım sunar. Tıpkı SAX gibi XML dosyasını parça parça okuyarak parse eden iterparse() methodu ve DOM gibi tek seferde yükleyen parse() yöntemi vardır.

3.YÖNTEM

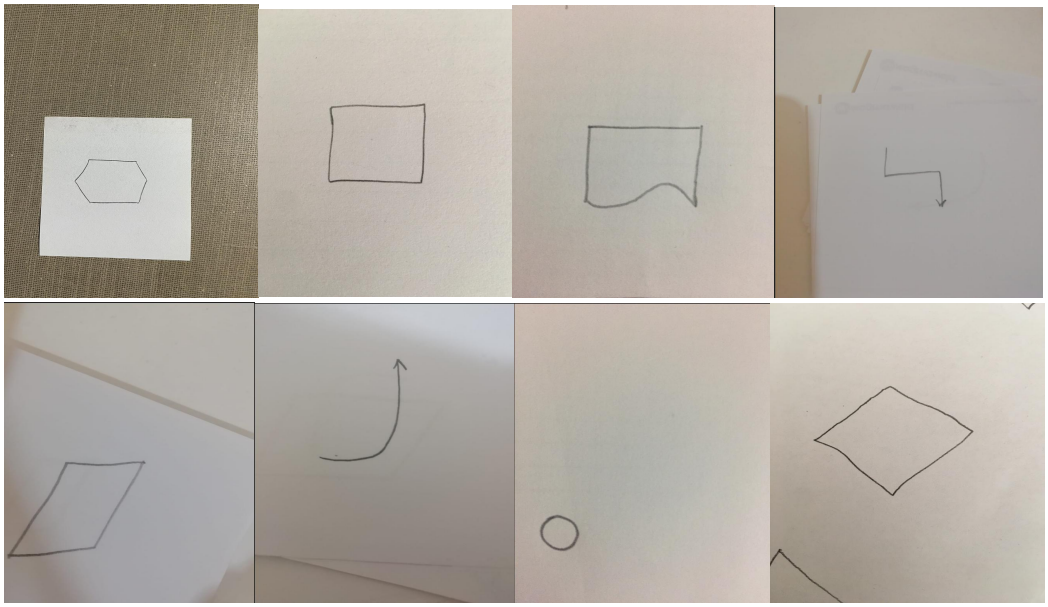
Bu bölümde yolo algoritması ile modelin eğitimi için gerekli veri setinin hazırlanması, modelin eğitim süreci, eğitimler ardından elde edilen sonuçlara göre yapılan geliştirmeler ve dijitalleştirme için XML koda dönüştürme adımlarının detaylarına yer verilmiştir.

3.1. Veri Setinin Hazırlanması

Proje kapsamında dijitale aktarılması planlanan akış diyagramı için belirlenen sembollere ait yeterli görüntü internet kaynaklarından elde edilemediği için veri seti oluşturma kararı alındı. Bunun üzerine proje ekibindeki herkes kendi elleriyle kağıtlara çizimler gerçekleştirdi ve çizimlerin resimlerini çekip tek bir yerde topladık. Veri setimizin içerisinde 6 sembol ve oklardan oluşmak üzere toplam da 7 şekil bulunmaktadır. Proje ekibi içerisindeki 6 kişi her sembolden 20 şer tane olmak kaydı ile toplamda 120 adet sembol ve 20 adette ok çizmişlerdir. Böylelikle her kişi 140 tane şekli farklı boyutlarda ve farklı yönlerde düz veya yamuk olacak şekilde düz beyaz kağıt üzerine el yordamı ile çizip veri setimiz oluşturulmuştur. Çizilen her bir şeklin fotoğrafı farklı açılardan, yakından ya da uzaktan çekilerek kaydedilmiştir.

Aşağıdaki resimlerde de görüldüğü üzere veri setimizi oluşturan şekiller elips, paralel kenar, altıgen, eşkenar dörtgen, belge, dikdörtgen ve oklardan meydana gelmektedir. Projeye başlarken özellikle seçilen sembollerin nedeni internet üzerindeki örnek akış diyagramlarının içeriğinde çoğunlukla bu 6 sembolün yer almasıdır.

Akış diyagramlarında elipsler başlangıç ve bitiş yerlerini, paralel kenarlar programa veri girişi yapılacağını, altıgen döngü olacağını, eşkenar dörtgen karar verme işlemi denilen (if-else) blok yapısını,belge ekrana veya yazıcıya bilgi çıkışı olacağını, dikdörtgenler aritmetik işlemleri veya herhangi bir atama işlemi yapılacağını ve son olarakta oklarımız diyagramın akış yönünü herhangi bir adımdaki işlem tamamlandıktan sonra hangi adıma gidileceğini göstermektedir.



Şekil 3.1.1 Veri Setinden Örnek Görseller

3.2. Eğitim Süreci

Oluşturulan veri seti kullanılarak yolo modelinin eğitilmesi için AlexeyAB tarafından geliştirilen darknet projesi kullanılmış ve eğitim platformu olarak google hizmetlerinden yararlanılması planlanmıştır. İlk olarak veri seti ve darknet dosyaları Google Drive üzerine yüklenmiş ve Colab kullanılarak çalıştırılmıştır. Temelinde CNN algoritması barındıran YOLO'nun hızlı çalıştırılması için Colab [13] servisinin sunmuş olduğu GPU özelliği aktifleştirilmiştir. Bunun için [14]darknetin kurulum dosyasında (makefile) şu değişiklikler yapılmıştır:

GPU=1

CUDNN=1

CUDNN_HALF=0

OPENCV=1

Bu ayarlamanın ardından :

darknet.exe detector train custom_data/labelled_data.data darknet/cfg/yolov3_custom.cfg backup/darknet53.conv.74 -dont_show

kodu çalıştırılarak eğitim başlatılmıştır. Bu kodun bileşenleri ve bu dosyalarda yapılan değişiklikler ise şu şekildedir:

- labelled_data.data dosyası train komutu için gerekli dosyaların konumlarını ve gerekli bilgileri tutar.Bunlar;

classes = 7

train = custom_data/train.txt

valid = custom_data/test.txt

names = custom_data/classes.names

backup = backup

şeklindedir.

- yolov3_custom.cfg dosyası yolo algoritmasının konfigürasyon dosyasıdır. Bu dosya algoritmanın parametrelerini belirler. Değişiklik yapılan satırlar şu şekildedir;

…

Training

batch=128

subdivisions=64

…

max_batches = 14000

steps=11200,12600

…

filters=36

…

Buradaki “batch” değişkeni tek iterasyonda belleğe yüklenecek fotoğraf sayısını , “subdivision” değişkeni yüklenen fotoğrafların kaç gruba ayrılacağını (batch/subdivison),

“max_batches” değişkeni toplam iterasyon sayısını belirtmektedir. “max_batches, steps, filters” değişkenleri AlexeyAB nin hazırladığı [18] dokümandaki bilgilere göre güncellenmiştir. Bu dokümanda maksimum iterasyon (sınıf sayısı * 2000) , adım sayısı maksimum iterasyonun %80 ve %90 değerleri , filtre ise (sınıf sayısı + 5) *3 şeklinde belirtilmiştir.

Bu değişikliklerin ardından eğitim başlatılmış ancak Colab’ın günlük kullanım süreleri nedeniyle eğitimin yarıda kesilmesi ve eğitim sürecinin bölünmesi nedeniyle modelin eğitim süreci bilgisayar ortamına aktarılmıştır. Modelin bilgisayar ortamında eğitilmesi için mevcut şartlar ve gereklilikler şu şekildedir:

- Nvidia GTX-1650 ekran kartı (mevcut kart)
- Nvidia CUDA Toolkit V11.3 [15]
- Cudnn V7.6.5 [16]
- OpenCV V4.5.2 [17]

Gerekli kurulumlar yapıldıktan sonra veri seti ile eğitim yapılmış ve on bin iterasyona kadar eğitime devam edilmiştir.

3.3. Test ve Geliştirme

Eğitilmiş model ile aşağıdaki kod kullanılarak görüntü içerisindeki nesnelere ait y çıktıları alınır , ayrıştırılır ve YOLO algoritmasında bahsedilen üst üste çizilmiş çerçeveler için maksimum olmayanı bastırma uygulanır.

Tasarlanan algoritma şu şekilde işlemektedir;

layers değişkenine modelin katmanlarını al

çıktı katmanlarını output layer değişkenine al

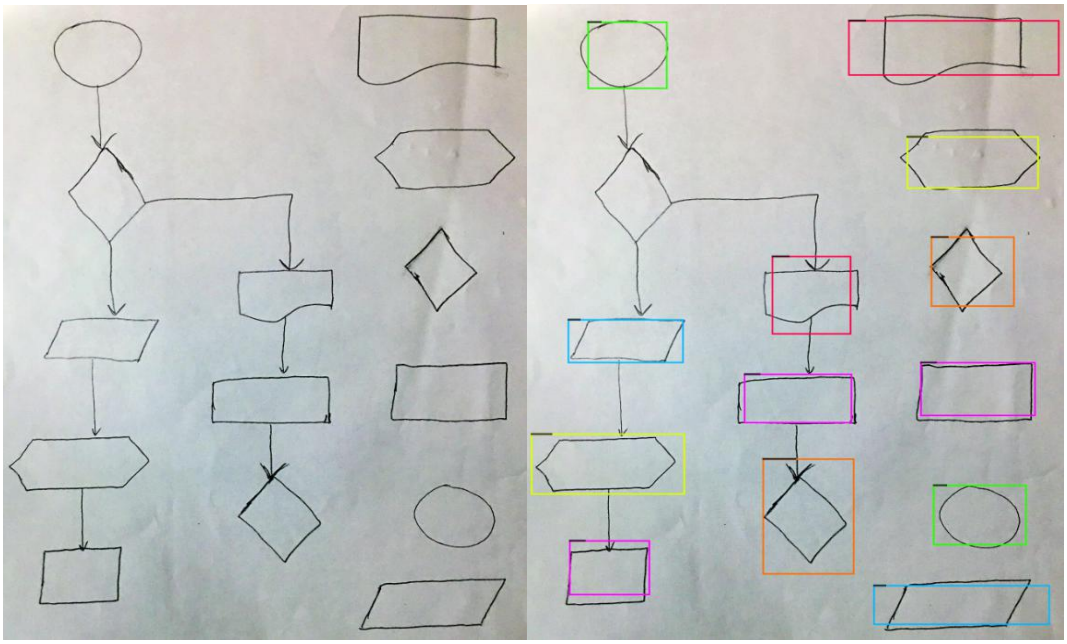
detection layer değişkenine çıktı katmanlarındaki matrisleri model üzerinden gönder

detection layer içindeki matrisleri gez

scorları bounding box haricinden itibariyle al

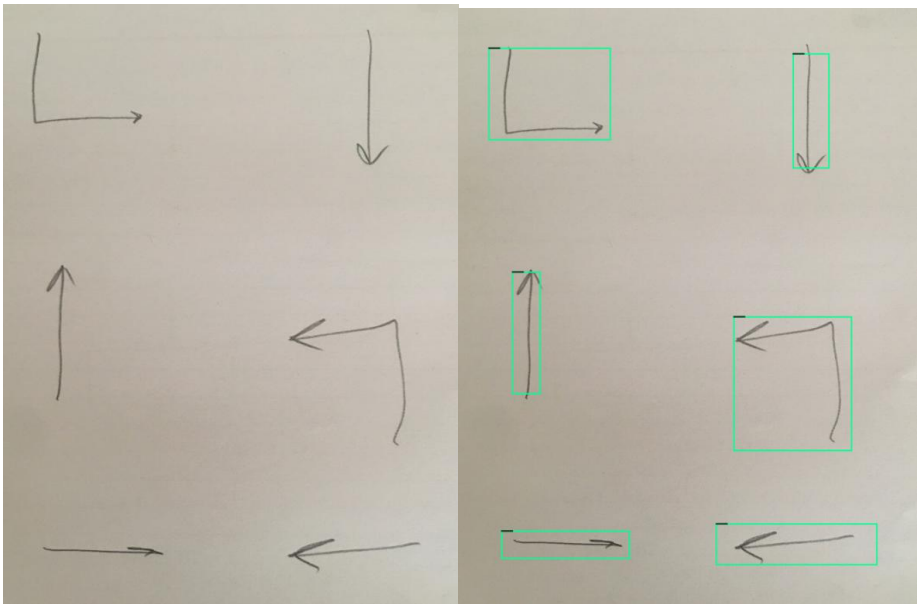
scoru en büyük olanın idsini predicted_idye ata
confidence değerine skor arasında en büyük olanı ata
confidence yüzde 20den büyük ise
label değişkenine predicted idye karşılık gelen etiketi ata
bounding boxın uzunluk genişlik ve merkez koordinatlarını al
start_x = int(box_center_x - (box_width/2))
start_y = int(box_center_y - (box_height/2))
bounding box değerleriyle başlangıç ve bitiş koordinatlarını al
nms metoduyla en yüksek güvenilirliğe sahip iç içe çizilen dikdörtgenlerin idlerini döndür
max ids içinde gez
max id'nin birinci parametresini max_class_id'ye ata
max_class_id'ye karşılık gelen kutuyu box_list'ten al ve box değişkenine ata
start_x = box[0]
start_y = box[1]
box_width = box[2]
box_height = box[3]
max_class_id'ye karşılık gelen kutuyu ids_list'ten al ve predicted_id değişkenine ata
label değişkenine predicted idye karşılık gelen etiketi labels listesinden ata
confidence değişkenine confidences_list'ten max_class_id'ye karşılık gelen confidenceyi ata

Eğitim sonucunda el çizimi diyagram ile test yapılmış ve aşağıdaki sonuç elde edilmiştir.



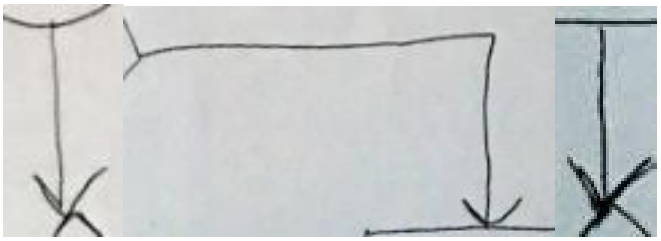
Şekil 3.3.1 On Bin İterasyon Test Sonucu

Tahmin edilmiş çıktı görüntüsü incelendiğinde karar (decision) sınıfından bir nesnenin ve okların tespit edilemediği görülmüştür. Öncelikle bu iki sınıfa ait veri setine ekleme yapılmış ancak yeniden eğitim sonucunda iyi bir sonuç alınamamıştır. Detaylı incelendiğinde veri setinde yer alan sınıflara ait görsellerin ayrı olarak çizilmesinin sorunun ana kaynağı olabileceği düşünülmüş bu nedenle sadece ok içeren ayrı bir fotoğraf ile test işlemi gerçekleştirilmiştir.



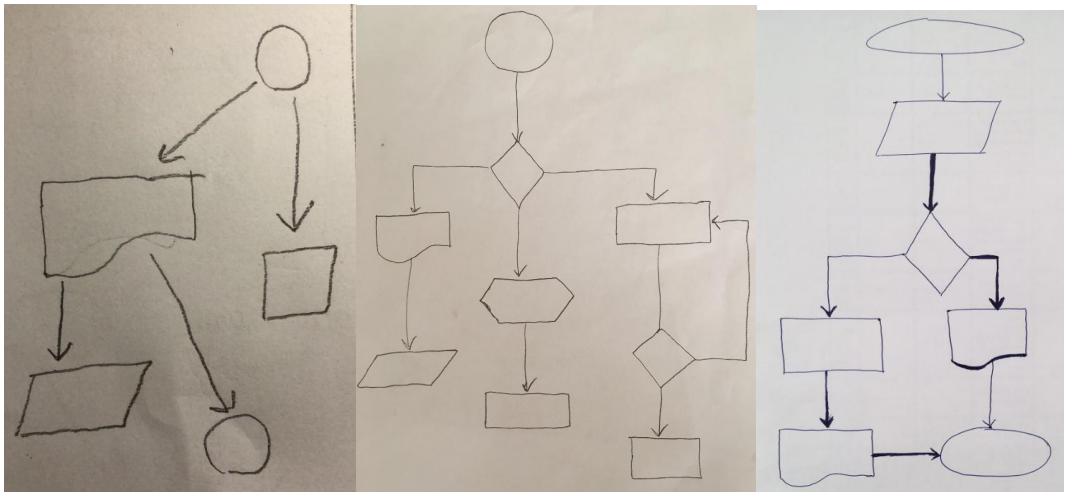
Şekil 3.3.2 Sadece Ok Nesnesi İçeren Test Görüntüsü ve Tahmin Edilmiş Çerçeveler

Ancak test görüntüsü için verilen görseldeki oklar incelendiğinde okların başlangıç ve bitiş yerlerinde diğer nesnelere ait çizgilerin yer aldığı görülmektedir.



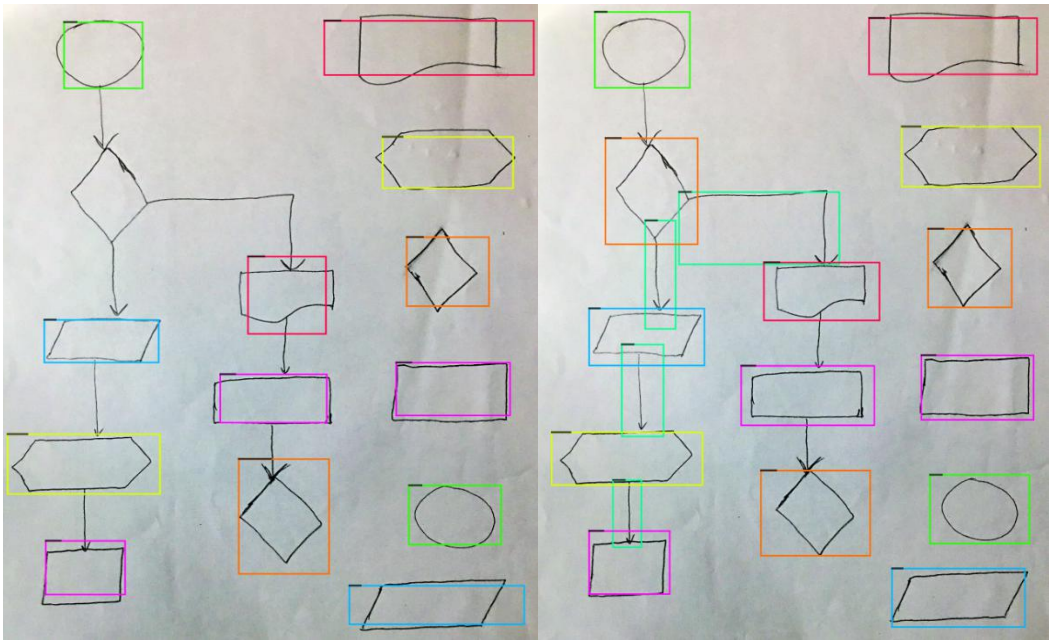
Şekil 3.3.3 Diyagram Örneği Üzerindeki Oklar

Bu sorunun çözümü için test görüntüsünde kullanılacak örnek diyagramlara benzer diyagramların, içerdikleri tüm nesneler etiketlenerek veri setine eklenmesi kararlaştırılmıştır.



Şekil 3.3.4 Veri Setine Eklenen Görseller

3 farklı kişi tarafından çizilmiş 40 diyagram görüntüsü veri setine eklenmiştir. Bu ekleme ile veri seti 880 görselden oluşmaktadır. Eklenen bu yeni görseller içerisinde toplamda onlarca nesne oklar ile birlikte kullanılmış ve her nesne etiketlenmiştir. YOLO modelinin eğitimi kaldığı yerden (10bin iterasyon) devam ettirilmiş ve 14 bin iterasyona tamamlanmıştır. Hesaplanan son ağırlık verisi kullanılarak aynı test görseli üzerinden elde edilen sonucun bir önceki sonuçla kıyaslaması şu şekildedir:

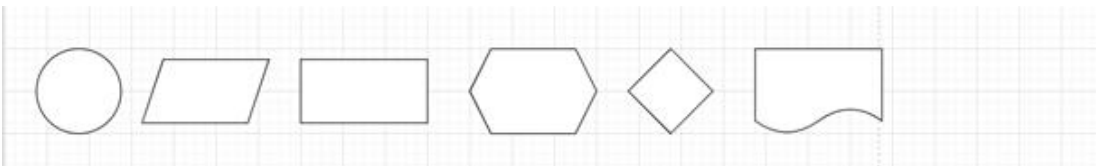


Şekil 3.3.5 Eski Test Sonucu (Sol) ve Yeni Test Sonucu (Sağ)

Şekil 3.3.5 de görüldüğü üzere yeni eklenen görseller ile karar (decision) sınıfına ait diğer nesne ile ok sınıfına ait nesnelerin büyük bir kısmı tespit edilebilmiştir.

3.4. Xml Formatına Dönüştürme

Nesneleri tespit etme işlemi [19] gerçekleştirildikten sonra bu nesnelerin çevrelendiği kutucuğa ait başlama ve bitiş konumlarını elde ettik. Elde edilen bu konumları Drawio'nun kabul ettiği xml formatında sıfırdan yazıldı ve bu xml formatındaki dosya bilgisayar ortamına çıkartıldı. Böylelikle Drawio'nun da kabul ettiği xml formatlı dosya Drawio içerisinde açılabildi. Nesneler xml formatında yazılmadan önce Drawio'nun içerisinde çeşitli semboller, oklar ile birlikte rastgele bir akış diyagramı çizildi. Çizilen bu rastgele akış diyagramına göre Drawio içerisinde bilgisayar ortamına uygulamanın kabul ettiği formatta xml dosyası çıkartıldı. Çıkartılan dosyanın incelenmesi sonucunda Drawio'nun kabul ettiği şablon belirlendi. Belirlenen bu şablona göre tespit ettiğimiz nesneleri uygulamanın xml formatında yazıldı.



Şekil 3.4.1 Uygulama üzerinden çizilen örnek şekiller

Şekil 3.4.2 Örnek Çizilen Şekillerin Xml formatı

İşlem (Process) şeklinin xml formatlı gösterimi için kullanılan kod şu şekildedir:

```
if label == "Process":
   elif label == "Process":
      for child in myroot:
         print(child.tag)
         for child_x in child:
                  child_x[0].append((ET.fromstring('<mxCellid="'+str(item_id)+'"
                  parent="1" vertex="1" style="rounded=0;whiteSpace=wrap;html=1;"
                  value=""><mxGeometryas="geometry"height="'+str(end_y-
                  start_y)+'"width="'+str(end_x-
                  start_x)+'"y="'+str(start_y)+'"x="'+str(start_x)+'"/></mxCell>')))
      item_id=item_id+1
```

Kodda görüldüğü üzere şekillerin yani mxcellid'lerin root tag'inin içinde olduğu için for döngüleriyle mxGraphModel'in altındaki root'un child'ı olduğunu gördük ve bu child'a erişildi. Kısaca root tag'inin içine girilip nesnelerin xml format şekilleri yazdırılacak.

Child içerisine hangi şekilin girilmesi gerektigini belirtebiliriz örneğin process şekili için yani dikdörtgen için bu tag girilebilir [20].

Elimizde bulunan 7 şeklin her biri için koddaki gibi karşılaştırma yaparak hangi şeklin öncelikli sırayla çizildiğini kararlaştırıyor.

Xml formatlı yazımda şekillerin yükseklikleri, genişlikleri ve konum bilgileri şu şekilde tutulmaktadır:

```
<mxCell id="6" parent="1" vertex="1" style="rhombus;whiteSpace=wrap;html=1;" value="">
<mxGeometry as="geometry" height="80" width="80" y="40" x="590"/>
</mxCell>
```

Nesnelerin belirtilen konum x ve y değerleri yerine başlangıç x ve başlangıç y değerleri yerleştirildi.

Başlangıç x, başlangıç y ve bitiş x, bitiş y koordinatlarını da (bitiş x-başlangıç x) ile genişlik değerini, (bitiş_y-başlangıç_y) yükseklik değerleri bulunup xml dosyasına yerleştirildi.

Xml'e semboller eklendikten sonra sıra akış diyagramının bağlantılarını sağlayan nesneler arası kullanılan oklara geldi. Okları konumlarıyla tespit ettikten sonra çıktığı nesneler (kaynak sembol) ve vardığı nesneleri (hedef sembol) belirlemek gerekmektedir.

Aşağıdaki kodda örnek ok gösterimi için xml formatında yazılımı görülmektedir.

```
<mxCell id="9" value=""
style="edgeStyle=orthogonalEdgeStyle;rounded=0;orthogonalLoop=1;jettySize=auto;html=1;" edge="1" parent="1" source="8" target="4">
        <mxGeometry relative="1" as="geometry" />
    </mxCell>
```

Burada okun çizimi için lazım olan, kaynak (source) ve hedef (target) nesnelerin sıralama Cell id'leridir. Bu elde edilen nesnelerin Cell id'lerine göre okların başladığı ve bittiği konumları belirlenmektedir.

Burda bahsi geçen kaynak ve hedef Cell id'lerini bulabilmek için her şeklin merkez noktasının koordinatlarını hesaplamak gerekmektedir. Bu hesaplamayı yaparken nesnelerin koordinatları olan başlangıç x ve y değerleri ile bitiş x ve y değerleri kullanıldı. (başlangıç_x+bitiş_x)/2 ile nesnenin merkez noktasının x koordinatı bulundu, (başlangıç_y+bitiş_y)/2 ile de nesnenin merkez noktasının y koordinatı bulunur.

Bulunan her nesnenin merkezi noktası bir listeye atıldı daha sonra bu listedeki koordinatlar ile çizilecek olan okun merkezi koordinatları arasındaki mesafe bulundu ve bu mesafeyi bulurken Formül (3.4.1)'den yararlanıldı.

$$|AB| = \sqrt{((x_2 - x_1)^2 + (y_2 - y_1)^2)}$$

(3.4.1)

```
def find_difference (arrow_center_x,arrow_center_y,box_center _list _x ,
box_center_list_y):                                                    difference_list=[]
                                                                      for x in
range(0,len(box_center_list_x)): difference=math.sqrt((((box_center_list_x[x]-
arrow_center_x)(box_center_list_x[x]-arrow_center_x))+((box_center_list_y[x]-
arrow_center_y)(box_center_list_y[x]-arrow_center_y))))
 difference_list.append(difference)
minimum=difference_list[0]
minimum_index=0
        secondminimum=difference_list[1]
        second_minimum_index=1
          n =len(difference_list)


for i in range(2,n):
    if difference_list[i]<minimum:
          secondminimum=minimum
          second_minimum_index=minimum_index
          minimum=difference_list[i]                    minimum_index=i
                                                    elif
          difference_list[i]<secondminimum and minimum !=difference_list[i]:
secondminimum=difference_list[i] second_minimum_index=i else: if
secondminimum == minimum: secondminimum = difference_list[i]
second_minimum_index=i
for i in range(0,len(difference_list)):
        print(difference_list[i])
return minimum_index,second_minimum_index
```

Yukarıdaki kodda okun merkezine en yakın 2 tane şeklin indexini döndürmektedir.

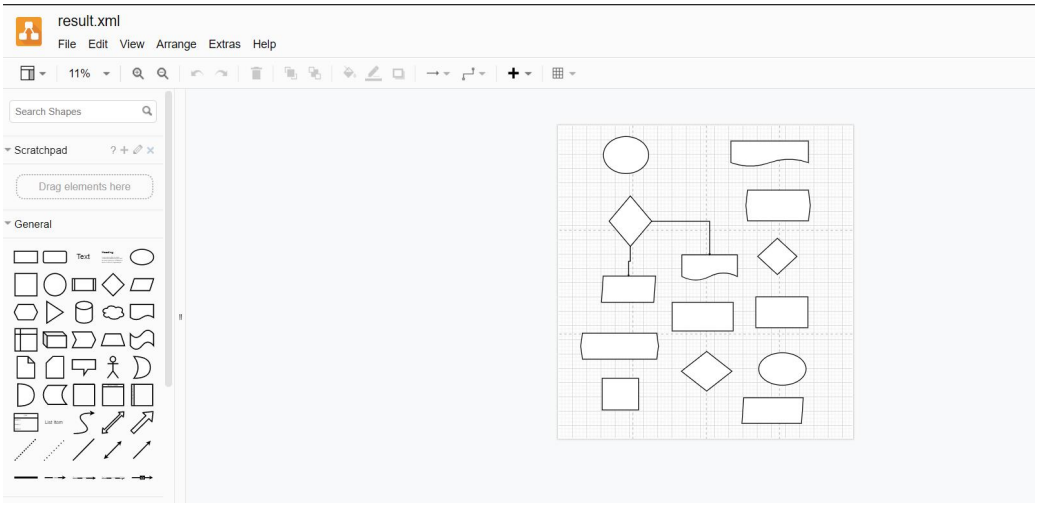
Okun merkez koordinatları ile diğer şekillerin merkez koordinatları arasındaki her mesafeyi ayrı bir listeye atıp bu liste içerisindeki değeri en küçük olan 2 değerin indeksini aldık. Bu indekslerin 2 fazlası bizim kaynak ve hedef nesnelerimizin temsil etmektedir. Burada 2 fazlası kullanılmasının sebebi ise şekillerin id'leri xml'de 2'den itibaren başlamalarıdır [21].

En yakın seçilen 2 uzaklığa ait nesnelerden y koordinatı küçük olan otomatik olarak diğerine göre diyagram içerisinde üst kısımda kalacağı için y'si küçük olan nesneyi kaynak (source) nesne diğer 2'nci küçük olan nesne de hedef (target) nesne olarak seçildi.

```
id_source=0
id_target=0
if box_center_list_y[index1]<box_center_list_y[index2]:
   id_source=index1+2
   id_target=index2+2
else:
   id_source=index2+2
   id_target=index1+2
child_x[0].append((ET.fromstring('<mxCell id="'+str(item_id)+'"
value="" style=" edgeStyle=orthogonal Edge Style; rounded=0; orthogonal Loop=1;
jettySize=aut o;html=1;"edge=”1”parent=”1”source="'+str(id_source)+'" target="'
+str(id_target)+'"> <mxGeometry relative="1" as="geometry" /></mxCell>')))
```

Yukarıdaki kodda kaynak ve nesnelerin xml yazılımını görmekteyiz.

Kullanılan test çizimi model ile test edilmiş, tespit edilen nesneler XML formatına çevrilmiştir. Elde edilen XML formatındaki belge draw.io platformunda açıldığında aşağıdaki çıktı elde edilmiştir.



Şekil 3.4.3 Akış Diyagramının Tespiti Sonrası Uygulamada Dijitalleştirilmesi


```
<?xml version="1.0"?>
- <mxfile etag="dGxDvoX6-5wLI0lh1zXa" version="14.6.11" agent="5.0 (Windows NT 10.0; Win64; x64) AppleWebKit/537.36 (KHTML, like Gecko) Chrome/90.0.4430.93 Safari/537.36" modified="2021-05-10T11:55:52.355Z" host="app.diagrams.net">
  - <diagram id="-QrMc-GOJyuFEOSV-_BC">
    - <mxGraphModel shadow="0" math="0" pageHeight="1169" pageWidth="827" pageScale="1" page="1" fold="1" arrows="1" connect="1" tooltips="1" guides="1" gridSize="10" grid="1" dy="491" dx="1038">
      - <root>
          <mxCell id="0"/>
          <mxCell id="1" parent="0"/>
        - <mxCell id="2" parent="1" value="" style="shape=parallelogram;perimeter=parallelogramPerimeter;whiteSpace=wrap;html=1;fixedSize=1;" vertex="1">
            <mxGeometry x="2057" y="3050" width="691" height="287" as="geometry"/>
          </mxCell>
        - <mxCell id="3" parent="1" value="" style="rounded=0;whiteSpace=wrap;html=1;" vertex="1">
            <mxGeometry x="2214" y="1919" width="586" height="346" as="geometry"/>
          </mxCell>
        - <mxCell id="4" parent="1" value="" style="ellipse;whiteSpace=wrap;html=1;aspect=fixed;" vertex="1">
            <mxGeometry x="2246" y="2546" width="533" height="363" as="geometry"/>
          </mxCell>
        - <mxCell id="5" parent="1" value="" style="shape=hexagon;perimeter=hexagonPerimeter2;whiteSpace=wrap;html=1;fixedSize=1;" vertex="1">
            <mxGeometry x="2101" y="723" width="726" height="344" as="geometry"/>
          </mxCell>
        - <mxCell id="6" parent="1" value="" style="rounded=0;whiteSpace=wrap;html=1;" vertex="1">
            <mxGeometry x="482" y="2832" width="410" height="355" as="geometry"/>
          </mxCell>
        - <mxCell id="7" parent="1" value="" style="rhombus;whiteSpace=wrap;html=1;" vertex="1">
            <mxGeometry x="1375" y="2531" width="570" height="444" as="geometry"/>
          </mxCell>
        - <mxCell id="8" parent="1" value="" style="shape=document;whiteSpace=wrap;html=1;boundedLbl=1;" vertex="1">
            <mxGeometry x="1380" y="1448" width="626" height="287" as="geometry"/>
          </mxCell>
        - <mxCell id="9" parent="1" value="" style="shape=document;whiteSpace=wrap;html=1;boundedLbl=1;" vertex="1">
            <mxGeometry x="1933" y="173" width="873" height="285" as="geometry"/>
          </mxCell>
        - <mxCell id="10" parent="1" value="" style="ellipse;whiteSpace=wrap;html=1;aspect=fixed;" vertex="1">
            <mxGeometry x="496" y="123" width="510" height="414" as="geometry"/>
          </mxCell>
        - <mxCell id="11" parent="1" value="" style="shape=hexagon;perimeter=hexagonPerimeter2;whiteSpace=wrap;html=1;fixedSize=1;" vertex="1">
            <mxGeometry x="240" y="2326" width="827" height="291" as="geometry"/>
          </mxCell>
        - <mxCell id="12" parent="1" value="" style="rhombus;whiteSpace=wrap;html=1;" vertex="1">
            <mxGeometry x="560" y="788" width="480" height="566" as="geometry"/>
          </mxCell>
        - <mxCell id="13" parent="1" value="" style="shape=parallelogram;perimeter=parallelogramPerimeter;whiteSpace=wrap;html=1;fixedSize=1;" vertex="1">
            <mxGeometry x="474" y="1686" width="610" height="294" as="geometry"/>
          </mxCell>
        - <mxCell id="14" parent="1" value="" style="rounded=0;whiteSpace=wrap;html=1;" vertex="1">
            <mxGeometry x="1271" y="1980" width="687" height="322" as="geometry"/>
          </mxCell>
        - <mxCell id="15" parent="1" value="" style="rhombus;whiteSpace=wrap;html=1;" vertex="1">
            <mxGeometry x="2235" y="1263" width="442" height="406" as="geometry"/>
          </mxCell>
        - <mxCell id="16" parent="1" value="" style="edgeStyle=orthogonalEdgeStyle;rounded=0;orthogonalLoop=1;jettySize=auto;html=1;" target="13" source="12" edge="1">
            <mxGeometry as="geometry" relative="1"/>
          </mxCell>
        - <mxCell id="17" parent="1" value="" style="edgeStyle=orthogonalEdgeStyle;rounded=0;orthogonalLoop=1;jettySize=auto;html=1;" target="8" source="12" edge="1">
            <mxGeometry as="geometry" relative="1"/>
          </mxCell>
        </root>
      </mxGraphModel>
    </diagram>
</mxfile>
```

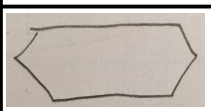
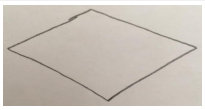
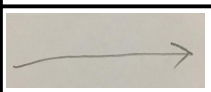

Şekil 3.4.4 Dijitalleştirilen Akış Diyagramının Xml Formatı

Şekil 3.4.5 , test için örnek verilen Şekil 3.3.1 deki diyagramda tespit edilen nesnelerin XML formatına ait çıktıdır. Bu XML dosyası draw.io platformunda açıldığında el çizimi olarak verilen görselin dijital çıktısı elde edilmektedir.

4. SONUÇLAR VE ÖNERİLER

Projenin amacı olan el çizimi akış diyagramlarının dijitale aktarılması hedefi kısmi olarak gerçekleştirilebilmiştir.

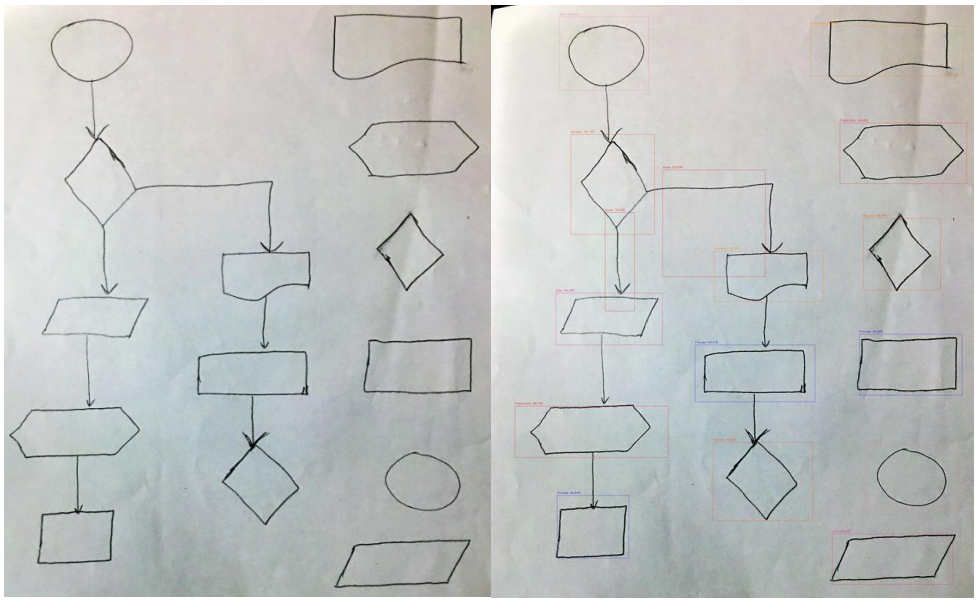
Tablo 4.1. Yapılan testler sonucu sınıfların tespit edilme oranları

	Şekil	Toplam	Tespit Edilen	Yüzde (%)
İşlem Sınıfı		32	31	96
Veri Sınıfı		23	23	100
Doküman Sınıfı		19	18	94
Hazırlık Sınıfı		19	19	100
Başlangıç-Bitiş Sınıfı		27	27	100
Karar Sınıfı		19	19	100
Ok Sınıfı		137	78	56

Tablo 4.1 de yer alan veriler, 3 farklı kişi tarafından oluşturulan akış diyagramlarında yer alan nesnelerin tespit edilme miktarlarını göstermektedir. Toplam sütunu test görsellerinde yer alan ilgili sınıfa ait toplam nesne sayısını , tespit edilen sütunu ise bu nesnelerden kaç tanesinin tespit edildiğini göstermektedir. Yüzde sütununda ise bu değerlerin yüzdelik başarı oranına yer verilmiştir. Ok sınıfı haricinde yüksek başarı oranı yakalanmıştır.

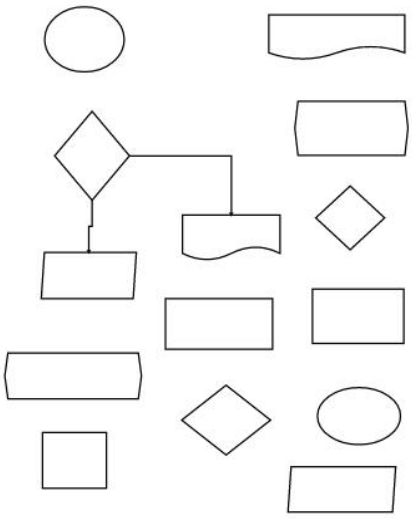
Mevcut aşamada çizilmiş akış diyagramına ait görüntü sisteme yüklenir , ardından program çıktı olarak tespit ettiği nesnelerin etrafına çerçeve çizerek tahmin ettiği sınıf isimlerini çerçeve üzerine yerleştirir.Tespit edilen nesneler için XML kod dönüşümü gerçekleşir ve hedef dosya yoluna XML uzantılı çıktı dosyası oluşturulur. Bu çıktı dosyası hedef platform olarak belirlenen draw.io üzerinde açıldığında tespit edilmiş nesnelerin platformun tanıdığı nesneler olarak görünmesi sağlanmaktadır.

Sonucun test edilmesi ve programın çalışma şeklinin anlatımı için örnek bir senaryo yürütülmüş ve aşağıda yer alan adımlarda anlatılmıştır.



Şekil 4.1 örnek diyagram çizimi (sol) ve çerçeveleri belirlenmiş hali (sağ)

Birinci adımda Şekil 4.1 de yer alan soldaki görselde akış diyagramında yer alan nesneler eğitilen model ile tespit edilmiş ve sağdaki çıktı elde edilmiştir. Bu çıktıda hangi nesnenin hangi koordinatlarda yer aldığı bilgisi elde edilmiş durumdadır. Ardından bu veriler XML dönüşüm aşamasında anlatıldığı gibi XMl formatına akarılmış ve uygun platformda açılarak aşağıdaki çıktı elde edilmiştir.



Şekil 4.2 draw.io da elde edilen çıktı görseli

Bu çıktıda da görülmektedir ki projenin temelinde yer alan şekillerden yalnızca ok nesnesi için tespit sıkıntısı yaşanmaktadır. Tespit edilebilen nesnelerin dijitalleştirilmesi başarı ile gerçekleştirilmiştir.

Projenin ilerletilmesi durumunda, daha fazla sınıf eklemesi yapılarak mevcut modelin üzerine yapılacak eğitim ile geliştirme yapılabilir. Ayrıca tanınan nesnelerin konumları elde edildiğinden diyagram nesneleri içerisinde yer alan yazıların da tespit edilebilmesi için sisteme ayrı bir yazı tanıma projesi entegre edilebilir.YOLO algoritması diyagram nesnelerinin ayrılması için yeterli bir algoritma olsa da yazı tanımak için farklı bir yönteme ihtiyaç duyulmaktadır.

KAYNAKÇA

[1] R. J. Rossheim, "Report on proposed American standard flowchart symbols for information processing", *Commun. ACM*, c. 6, sy 10, ss. 599-604, Eki. 1963, doi: 10.1145/367651.367657.

[2] B. Schafer ve H. Stuckenschmidt, "Arrow R-CNN for Flowchart Recognition", içinde *2019 International Conference on Document Analysis and Recognition Workshops (ICDARW)*, Sydney, Australia, Eyl. 2019, ss. 7-13. doi: 10.1109/ICDARW.2019.00007.

[3] C. Xiao, C. Wang, ve L. Zhang, "PPTLens: Create Digital Objects with Sketch Images", içinde *Proceedings of the 23rd ACM international conference on Multimedia*, Brisbane Australia, Eki. 2015, ss. 745-746. doi: 10.1145/2733373.2807974.

[4] J. Wu, C. Wang, L. Zhang, ve Y. Rui, "Offline Sketch Parsing via Shapeness Estimation", s. 7.

[5] B. Plimmer ve I. Freeman, "A Toolkit Approach to Sketched Diagram Recognition", program adı: Proceedings of HCI 2007 The 21st British HCI Group Annual Conference University of Lancaster, UK, Eyl. 2007. doi: 10.14236/ewic/HCI2007.21.

[6] S. Chakraborty, S. Paul, ve Sk. Md. Masudul Ahsan, "A Novel Approach to Rapidly Generate Document from Hand Drawn Flowcharts", içinde *2020 IEEE Region 10 Symposium (TENSYMP)*, Dhaka, Bangladesh, 2020, ss. 702-705. doi: 10.1109/TENSYMP50017.2020.9231033.

[7] J. Du, "Understanding of Object Detection Based on CNN Family and YOLO", *Journal of Physics*, s. 9.

[8] J. Redmon ve A. Farhadi, "YOLO9000: Better, Faster, Stronger", *arXiv:1612.08242 [cs]*, Ara. 2016, Erişim: May. 19, 2021. [Çevrimiçi]. Erişim adresi: http://arxiv.org/abs/1612.08242

[9] J. Redmon ve A. Farhadi, "YOLOv3: An Incremental Improvement", s. 6.

[10]Y. Mesci, "YOLO Algoritmasını Anlamak", Medium, Kas. 23, 2019. https://medium.com/deep-learning-turkiye/yolo-algoritmas%C4%B1n%C4%B1-anlamak-290f2152808f (erişim May. 19, 2021).

[11] "Yolo Framework | Object Detection Using Yolo", *Analytics Vidhya*, Ara. 06, 2018.https://www.analyticsvidhya.com/blog/2018/12/practical-guide-object-detection-yolo-framewor-python/ (erişim May. 19, 2021).

[12] A. Neubeck ve L. Van Gool, "Efficient Non-Maximum Suppression", içinde *18th International Conference on Pattern Recognition (ICPR'06)*, Ağu. 2006, c. 3, ss. 850-855. doi: 10.1109/ICPR.2006.479.

[13] “Google Colaboratory”.https://colab.research.google.com/notebooks/intro.ipynb (erişim May. 19, 2021).
[14] “YOLO: Real-Time Object Detection”. https://pjreddie.com/darknet/yolo/ (erişim May. 19, 2021).
[15] “CUDA Toolkit 11.3 Downloads”, *NVIDIA Developer*, Nis. 15, 2021. https://developer.nvidia.com/cuda-downloads (erişim May. 19, 2021).
[16]“NVIDIA cuDNN”, *NVIDIA Developer*, Eyl. 02, 2014. https://developer.nvidia.com/cudnn (erişim May. 19, 2021).
[17] “Releases”, *OpenCV*. https://opencv.org/releases/ (erişim May. 19, 2021).
[18]“GitHub - AlexeyAB/darknet: YOLOv4 / Scaled-YOLOv4 / YOLO - Neural Networks for Object Detection (Windows and Linux version of Darknet)”. https://github.com/AlexeyAB/darknet (erişim May. 19, 2021).
[19]“Bilgisayarlı Görü: YOLOv4 ile Nesne Tanıma | Udemy”. https://www.udemy.com/course/bilgisayarl-goru-yolov4-ile-nesne-tanma/?start=15 (erişim May. 19, 2021).
[20]“Extracting the XML from mxfiles - draw.io”. https://drawio-app.com/extracting-the-xml-from-mxfiles/ (erişim May. 19, 2021).
[21]“mxgraph - Format of Draw.io XML file? - Stack Overflow”. https://stackoverflow.com/questions/59416025/format-of-draw-io-xml-file (erişim May. 19, 2021).

EKLER

Ek1.

Bölüm 3.3'teki test aşaması için kullanılan kod;

“

```
layers = model.getLayerNames()
output_layer = [layers[layer[0]-1] for layer in model.getUnconnectedOutLayers()]

model.setInput(img_blob)
detection_layers = model.forward(output_layer)

ids_list = []
boxes_list = []
confidences_list = []

#%% 4. Bölüm

for detection_layer in detection_layers:
        for object_detection in detection_layer:

        scores = object_detection[5:]
        predicted_id = np.argmax(scores)
        confidence = scores[predicted_id]

        if confidence > 0.20:

        label = labels[predicted_id]
        bounding_box = object_detection[0:4] *
np.array([img_width,img_height,img_width,img_height])
        (box_center_x, box_center_y, box_width, box_height) =
bounding_box.astype("int")

        start_x = int(box_center_x - (box_width/2))
        start_y = int(box_center_y - (box_height/2))

        ids_list.append(predicted_id)
        confidences_list.append(float(confidence))
        boxes_list.append([start_x, start_y, int(box_width), int(box_height)])

list_start_x=[]
list_start_y=[]
list_end_x=[]
list_end_y=[]
```

```
max_ids = cv2.dnn.NMSBoxes(boxes_list, confidences_list, 0.5, 0.4)
item_id=2
for max_id in max_ids:

        max_class_id = max_id[0]
        box = boxes_list[max_class_id]

        start_x = box[0]
        start_y = box[1]
        box_width = box[2]
        box_height = box[3]

        predicted_id = ids_list[max_class_id]
        label = labels[predicted_id]
        confidence = confidences_list[max_class_id]
“
```

ÖZGEÇMİŞ

Aziz Yelbay 1999 yılında İstanbul'da doğdu.Lise eğitimini İstanbul'da Şehremini Anadolu Lisesi'nde tamamladı.2017 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği lisans bölümünü kazandı ve halen devam etmektedir. Java web teknolojileri ile ilgilenmektedir ve kendini geliştirme aşamasındadır.

Aziz Yelbay

Serkan Özdemir 1998 yılında İstanbul'da doğdu.İlkokul ve lise eğitimini İstanbul'da tamamladı.2017 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği bölümünü kazandı ve öğrenimine devam etmektedir.

Serkan Özdemir

Aziz Batuhan Bıyıklı 1999 yılında İstanbul'da doğdu. Lise eğitimini İstanbul'da Maltepe Fen Lisesi'nde tamamladı. 2017 yılında Kocaeli Üniversitesi Bilgisayar Mühendisliği lisans bölümünü kazandı ve halen devam etmekte. Java, gömülü sistem (embedded) teknolojileri ile ilgilenmekte ve kendini geliştirme aşamasında.

Aziz Batuhan Bıyıklı