ÇANAKKALE ONSEKİZ MART ÜNİVERSİTESİ
MÜHENDİSLİK FAKÜLTESİ
BİLGİSAYAR MÜHENDİSLİĞİ BÖLÜMÜ

MAKİNE ÖĞRENİMİNİN GÜNLÜK HAYATTA KULLANIMI

Baran AKÇAKAYA
Egemen İNCELER

Dr.Öğr.Üyesi AHMET CUMHUR KINACI

Ocak, 2021
ÇANAKKALE

MAKİNE ÖĞRENİMİNİN GÜNLÜK HAYATTA KULLANIMI

Çanakkale Onsekiz Mart Üniversitesi Mühendislik Fakültesi
Bitirme Ödevi
Bilgisayar Mühendisliği Bölümü

Baran AKÇAKAYA
Egemen İNCELER

Dr. Öğr.Üyesi AHMET CUMHUR KINACI

Ocak, 2021
ÇANAKKALE

Baran AKÇAKAYA ve Egemen İNCELER, tarafından **Dr. Öğr.Üyesi AHMET CUMHUR KINACI** yönetiminde hazırlanan **"Makine öğreniminin günlük hayatta kullanımı"** başlıklı çalışma tarafımızdan okunmuş, kapsamı ve niteliği açısından bir Bitirme Ödevi olarak kabul edilmiştir.

.....

Danışman

Bölüm Başkanı
Bilgisayar Mühendisliği Bölümü

TEŞEKKÜR

Bu projede bize güvenen ve her aşamada destek veren sayın Dr. Öğr.Üyesi AHMET CUMHUR KINACI'ya ve manevi olarak desteğini bizden asla esirgemeyen Metin BİLGİN'e teşekkür ederiz.

Baran AKÇAKAYA
Egemen İNCELER

SİMGELER VE KISALTMALAR

CPU (Central Processing Unit): Merkezi İşlem Birimi

MS (Milli Second): Mili Saniye

FPS (Frame Per Second): Saniye Başına Görüntü Karesi

RAM (Random Access Memory): Rastgele Bellek Erişimi

vCPU (virtual Central Processing Unit) : Sanal Merkezi İşlem Birimi

HD (High Definition): Yüksek Çözünürlüklü

MB: Megabyte

KB: Kilobyte

OS (Operating System): İşletim Sistemi

RT (Response Time): Cevap Süresi

GPU (Graphics Processing Unit): Grafik İşlemci Birimi

Faster-RCNN (Faster-Region Convolution Neural Network): Bölgesel Arama Sinir Ağı

MAKİNE ÖĞRENİMİNİN GÜNLÜK HAYATTA KULLANIMI

ÖZET

Bu projenin amacı; bir bilgisayarlı görü algoritması kullanarak android telefondan gerçek zamanlı nesneleri tespit edip, bu nesneler hakkında internetten veriler çekerek kullanıcı bilgilendirmek.

İÇİNDEKİLER

Sayfa

BÖLÜM 1
GİRİŞ

Projemizi belirledikten sonra, ilk olarak kendi veri setimizi hazırlayıp model eğitmek mi yoksa Aktarımlı Öğrenme (Transfer Learning) kullanmak mı sorusuna cevap aradık. Aktarımlı öğrenme, önceki bilgiler kullanılarak daha az eğitim verisi ile daha yüksek başarı gösteren ve daha hızlı öğrenen bir model eğitme yöntemidir. Yani aktarımlı öğrenme genelde düşük donanıma sahip makinelerde, makine öğrenimi veya yapay zekâ uygulamalarındaki başarıyı artırmak için kullanılır. Elimizdeki donanım gücü düşük olduğu için bizde bu projede bu yöntemi seçtik. Eğer bu yöntemi seçmeyip kendi modelimizi oluşturup eğitseydik çok uzun süren eğitim süresinden sonra bile az başarı oranı yakalayacaktık. Bunun haricinde modelimizi eğitmek için gerekli verileri dahi toplayamayacaktık.

Aktarımlı öğrenme yöntemi üzerinde karar kıldıktan sonra, bu yöntem ile hangi nesne tanıma modelini kullanacağımız üzerine araştırmalar yaptık. Bu araştırmalarda kullanabileceğimiz birçok nesne tanıma modeli bulduk. Bunlar:

VGG-16: Basit bir ağ modeli olup öncesindeki modellerden en önemli farkı evrişim katmalarının 2'li ya da 3'li kullanılmasıdır. İki FC (Full Connection) katmanı çıkışında 1000 sınıflı softmax başarımı hesaplanır. Yaklaşık 138 milyon parametre hesabı yapılmaktadır.

VGG-19: 5 ortaklama katmanı ve 3 tam bağlı katmanda sona eren 16 evrişim katmanından meydana gelen ve ReLU aktivasyon fonksiyonlarıyla tanımlı bir derin öğrenme modelidir.

ResNet-50: Resnet, artık değerli nöral ağların (residual neural network) bir kısaltmasıdır. Bu model evrişimli sinir ağlarının (convolutional neural network:CNN) geliştirilmiş bir versiyonudur. Resnet modeli CNN ağlarının performans düşümü (degradation) problemini çözmeyi hedeflemektedir.

Inception: Inception ağ modeli modüllerden oluşmaktadır. Her bir modül, farklı boyutlu evrişim ve max-pooling işlemlerinden meydana gelmektedir.

YOLOv3: YOLO, konvolüsyonel sinir ağlarını (CNN) kullanarak nesne tespiti yapan bir algoritmadır. YOLO algoritması çalışmaya başladığında görüntülerdeki veya videolardaki nesneleri ve bu nesnelerin koordinatlarını aynı anda tespit eder.

Yukarıda açıklanan nesne tanıma modelleri arasından YOLOv3 modelini seçtik. Bu seçimi yaparken, Google Colab üzerinde denemeler yaptık. Bu denemeleri yaparken tahmin süresi ve tek resimde birden fazla nesne tanıma gibi iki önemli noktayı dikkate aldık. Bunun sebebi gerçek zamanlı nesne tespiti yaparken tahmin süresinin çok önemli olması ve gelen resimde birden fazla nesne olabileceği için bu nesneleri tek seferde tahmin etmek çok önemlidir. Bu iki noktayı dikkate aldığımızda ResNet-50, VGG-16 ve VGG-19 modelleri çok daha fazla nesne çeşidi tespit etmesine rağmen hem tahmin süresi hem de resimlerde sadece bir tane nesne tespit ettiği için bu iki modeli eledik. Inception modeli ise 1000 ms'i aşan tahmin süresi olduğu için bu modeli de eledik. Yolov3 modeli, diğer nesne tanıma modellerine göre çok daha amacımıza uygun olduğu için bu modeli seçmeye karar verdik.

Modelimizi belirledikten sonra, günlük hayat şartlarını düşünerek mobil uygulama ile entegre etmeye karar verdik çünkü insanlar günlük hayatında telefonları ile çok vakit geçiriyor ve bu da pazarın mobil dünyaya kaymasını sağlıyor. Bu sayede daha çok insan ile etkileşime geçerek, bu alandaki gelişimin hızlanmasına katkı sağlıyor.

BÖLÜM 2
ANA BÖLÜM

Bildiğimiz gibi Yolov3 modelini ve android ortamını seçmiştik. İki farklı çalışma ortamını seçmemizin sebebi, bir sunucu-istemci modeli geliştirmek istememizdir. Çünkü Yolov3 modeli ağır bir model olduğu için bu modeli Android'e entegre etmemiz oldukça zor. Bizde bunu yapmak yerine android cihazımızın kamerasından aldığımız görüntüleri Yolov3 modelinin çalıştığı bir Python sunucusuna gönderip tahmin yaptırarak sonuçları tekrar android cihazına göndermeyi hedefledik. Bunun içinde ilk olarak Yolov3 modelini çalıştıracağımız sunucu ortamını belirledik. Bunun için öncelikle Google Colab'ı kullandık.

Google Colab, Google'ın desteklediği ve kullanıcılara donanın ihtiyacını ortadan kaldırarak uygulama geliştirme imkânı sağladığı bir platformdur ve Google Drive ile entegre çalışabilir. Biz de ilk olarak bu ücretsiz platform üzerinden çalışmaya başladık. Modelimizi Colab'a yükleyip çalıştırdığımızda 500-1000 ms arasında bir tahmin süresi olduğu gördük. Bu süre gerçek zamanlı nesne tespitinde çok uzun bir süre çünkü saniyede 20 fps aldığımız bir durumda bir resmin tahmininin minimum 500 ms olduğunu varsayarsak 20 resmi tahmin etmesi minimum 10000 ms olacaktır. Yani bir saniyelik resmi on saniyede tahmin etmiş olur.

Google Colab'da sürenin çok uzun olduğunu gördüğümüzde alternatif sunucu arayışına geçtik ve Amazon'un EC2 sunucusunu denemeye karar verdik. Amazon EC2, Amazon Web Services'ın bulut bilişim platformudur. Platform, kullanıcıların kendi bilgisayar uygulamalarını çalıştırabilecekleri sanal bilgisayarlar kiralamalarına olanak tanımaktadır. Ancak, Amazon EC2 sunucuları donanım özelliklerine göre değişen ücretlere tabii tutulur. Yalnız öğrenciler için Amazon, öğrencilik döneminde harcamaları için $30'lık bir kredi sağlamaktadır. Bizde bu öğrenci kredisini kullanarak sunucumuzu oluşturduk.

Amazon EC2'de bize sağlanan makineler boş olarak gelmektedir. Bu makinelerin içine birçok farklı işletim sistemi kurulabiliyor. Biz sunucumuza Ubuntu Server 20.04 LTS kurduk. Bu adımdan sonra sunucumuzun donanım özelliklerini (Exp: 16 RAM, 4 vCPU) belirledik. Birkaç adım daha sonra sunucumuzu kurduk. Bu makineye bağlanmak için ve android iletişimi sağlamak için SSH bağlantımızı

kurmamız gerekmektedir. Bu SSH bağlantısını Putty ile sağlıyoruz. Makine kuruurken bize bir '.ppk' uzantılı dosya indirmemizi istemekte ve bu dosyayı PuttyGen ile '.pem' dosyasına dönüştürerek sanal sunucumuza bağlanmaktayız. İşletim sistemi kurulduktan sonra makineye Python, Keras, Tensorflow, OpenCV, pip3 gibi modülleri yükledik.

Amazon EC2 makinemizdeki tüm kurulum işlemlerini tamamladıktan sonra Yolov3 modelimizi de sunucumuza yükledik. Bu model iki Python dosyasından oluşmaktadır. İlk dosyamızda Yolo ağından oluşan bir yapı bulunmaktadır. Bu yapıyı Yolo'nun sağladığı '**yolov3.weight**' dosyası ile eğitim yaparak '**model.h5**' dosyasına kaydediyoruz. İkinci Python dosyamızda ise eğittiğimiz bu '**model.h5**' dosyasını kullanarak tahminlerde bulunuyoruz. Sunucumuza ikinci Python kodunu ve '**model.h5**' dosyasını yükledik. Ayrıca test etmek için birkaç tane resim yükledik ve modelimizi sunucumuzda başarıyla çalıştırdık. Bu çalışmanın sonucunda 200 ms gibi bir tahmin süresine ulaştık. Bu da Google Colab performansının neredeyse 2-3 katıdır. Tüm bunlar göz önüne alındığında Amazon EC2 makinelerinde projemizi devam ettirmeye karar verdik.

Android cihazımızda, kamera entegrasyonundan sonra, ImageAnalysis modülünü kullanarak kameradan gelen frame'leri aldık. Server'a bu resimleri göndermek için yeniden boyutlandırma ve birkaç format değişikliği yaptık. Çünkü aldığımız resimler HD kalitesinde olduğu için resmin boyutu çok büyük oluyordu. Bu yüzdende resim verisini gönderirken gönderme süresi çok uzun oluyor. Zaten göndereceğimiz resmi yolov3 modeli yeniden boyutlandırma yapacağı için burada gönderdiğimiz resmin kalitesinin pek önemi yok. Bunun için 480x600 boyutlarında ve quality değerini 50 olarak ayarladık. Yeniden boyutlandırma işlemine ek olarak, kameradan gelen frameler yan geldiği için 90 derecelik sağ rotate ederek düzeltiyoruz. Tüm bu işlemler sonucunda göndereceğimiz resmin boyutu 2-3 mb aralığında değişen resim boyutu 100-300 kb arasına indirmeyi başardık. Bu işlemlerden sonra aldığımız resimlerde nesne tespiti yapacağımız için bu resimleri Python servera göndermemiz gerekiyor. Bu yüzden aldığımız resimleri direkt ekrana basamıyoruz. Bunun için kullanıcı ara yüzünde ImageView objesi oluşturarak gösteriyoruz. ImageAnalysis'den gelen frameleri Python serverına göndermek için socket yapısını kullanıyoruz. Kullandığımız kütüphane integer, bytearray ve charset

veri tiplerini desteklediği için, aldığımız resmi bytearray'e çevirerek gönderme işlemini yaptık. Yalnız, gönderdiğimiz resmin nerede sonlandığını belirtmek için **'\sended'** belirtecini bytearray'in sonuna ekledik. Serverdan gelen verilerimizi almak için de socket yapısını kullandık. Fakat Android OS, veri alma ve veri gönderme işlemlerini arka planda yapmamızı istiyor. Arka planda yapmak için ise Android servislerini kullanarak bir servis yazdık. Bu servis arka planda bir thread oluşturuyor ve veri alma işlemini gerçekleştiriyor. Tüm bunlar Android OS'un ana threadinde olmadığı için ve kullanıcı ara yüzüne buradan ulaşmamıza izin vermediği için tüm android uygulamalarından dinlenebilen ve mesaj yayınlayan BroadcastReceiver yazdık. Bu mesajı kullanıcı ara yüzünde gösterme işlemleri yapıldı.

Android cihazımız ile Python serverımız arasındaki bağlantıyı sağlamak için socket yapısını kullanmaya karar verdik. Bunun için serverdaki kodumuza TCP serverı entegre ettik. Ancak socket yapısı hataya çok müsaittir. Bu yapıyı kurarken dikkat olmalıyız. Bizde bu yapı ile uğraşırken çeşitli hatalar aldık. Android cihazımızdan servera gönderdiğimiz resimler format hatasından dolayı doğru bir şekilde kullanılamıyordu. Ayrıyeten servera seri bir şekilde veri gönderirken TCP sunucusuna verimizin nerede bittiği bilgisini de göndermeliyiz. Aksi takdirde bu bilgiyi göndermezsek sunucu verinin nerede bittiğini bilmediği ve sürekli veri geldiği için verileri bir bütün olarak alıyor ve biz bu verileri ayrıştıramıyoruz. Bunu engellemek için ise her veriden sonra bir ayırt edici ifade (Exp: '\sended', '\stop') eklenmeli. Bir diğer aldığımız büyük hata ise, android'de kamera frame'lerini alırken random bir yerde serverımıza boş veriler göndermekte ve server, boş veriler gelmeye devam ettiği için socket bağlantısını koparmayıp sürekli veri almaya devam ediyordu. Bu hatayı server koduna eklediğimiz bir boş değer kontrol komutu ve socket bağlantısını yeniden kurarak bu hatayı engelleyebiliriz.

Socket işlemlerini bitirdikten sonra, fps'i artırmak için thread ve process'leri kullanmaya başladık. Bu iki yapıyı da tek tek denedikten sonra senkronizasyon sorunu oluştuğunu fark ettik. Yani aynı anda gelen resimlerin tahmini sırasında, hangi thread ya da process'in daha önce biteceğini bilemediğimiz için bunların senkronizasyonu zaman alıcı oluyor. Ayrıyeten Amazon EC2 sunucumuz, öğrenci hesabı olduğu için bize sundukları makinelerin hiçbirinde ekran kartı desteği bulunmuyor. Bu yüzdende ekran kartlarında aynı anda birden fazla yolov3 modelini

yükleyip bunları ayrı ayrı çalıştıramıyoruz. Çalıştırsak dahi buradaki tahmin süresi minimum 4 kat artıyor. Bu da bizim sorunlarımızı çözebilen bir tahmin süresi değil. Bu şekilde çözemeyeceğimizi anladıktan sonra resimleri birleştirerek tek seferde tahmin etme yöntemini denedik. Bu yöntemde gelen dokuz resmi numpy kütüphanesi kullanarak birleştirme işlemini yapıyoruz. Daha sonrasında bu birleştirilen resimleri, tek bir resimmiş gibi modelimize verip tahmin etmesini sağlıyoruz. Normalde biz resimleri tahmin ettirirken yolov3'ün default 100x100 olarak yeniden boyutlandırmasını yaptırıyoruz. Ancak dokuz resmi birleştirdiğimizde bu defualt boyutu artırılması gerekmekte. Çünkü dokuz resimden oluşan birleştirilmiş resmin boyutu çok büyüdüğünden 100x100 formatında küçük ayrıntılar kayboluyordu. Bu yüzdende bizim birleştirdiğimiz resmin boyutuna göre defualt boyutu değiştirmeniz gerekmekte. Biz bu default boyutu 480x480 olarak değiştirdik.

Sunucumuzdan yeteri kadar performans alamadığımız için (Exp: rt = 600ms) local bilgisayarımızda deneme kararı aldık. Bunun için local bilgisayarımıza CUDA kurulumunu yaptık. CUDA, GPU için NVIDIA'nın sunduğu C programlama dili üzerinde eklenti olarak kullanıma sunulan bir mimari ve teknolojidir. GPU üzerinde çalışmasını sağlayan geliştirme araçları kümesidir. Bu sayede, tahmini GPU üzerinden yapmamızı sağladı. Gerekli kurulumları yaptıktan sonra tahmin süresinin 3 kat azaldığını gördük. Ayrıyeten bizim Amazon EC2 sunucumuz Amerika'da bulunmakta. Bu yüzden gönderdiğimiz ve aldığımız resimlerin gönderim süresi çok uzundur (Exp: 3000ms). Eğer bu işlemleri local bilgisayarımızda yaparsak 50 ms gibi çok kısa bir süreye kadar düşmekte. Bu da bizim local'e geçmemizin en büyük sebeplerinden biridir.

Bugüne kadar hep tahmin süresini esas almıştık. Daha sonrasında yolov3 modelinin resimleri tanımlayıp çerçeveleme süresinin, tahmin süresinden çok daha fazla olduğunu gördük. Bu yüzden server kodumuzda bazı değişiklikler yapmak zorunda kaldık. Yolov3, faster-RCNN yapısını kullandığı için ilk olarak resimde tahmini nesne bulanabilecek yerleri çerçeveliyor. Daha sonrasında bu çerçeve içindeki nesnelerin eşik değerini geçip geçmediğini kontrol ediyor. Her resim için 14000'den fazla çerçeve tanımlayan yolov3 modeli, toplam tahmin süresini 10 saniyenin üzerine çıkartıyor. Biz burada daha çerçeveyi oluştururken çerçevenin eşik değerini kontrol ediyoruz, geçen çerçeveleri bir listenin içine atıyoruz. Bu da her resim için 14000 olan çerçeve sayısını 10-30 arası bir miktara düşürüyor. Bu da tahmin süresini 15 kat hızlandırdı.

BÖLÜM 3

SONUÇ

Şu an projemizde birkaç problem bulunmakta. Bunlar, veri gönderip alırken senkronizasyon olmaması ve veri tipi farklılığından dolayı oluşan tip uyuşmazlığı sorunları bulunmakta. Bu iki problem dışında veri gönderip alırken hiçbir sorun bulunmamakta ve tahminleri başarılı bir şekilde yapmaktadır.

```
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: ]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
response: [254.0, 341.5, 'laptop', 97.1]
```

Şekil 1.1

Şekil 1.2

Şekil 1.3

KAYNAKLAR

- https://medium.com/nsistanbul/g%C3%B6r%C3%BCnt%C3%BC-tan%C4%B1yan-mobil-uygulama-nas%C4%B1l-geli%C5%9Ftirilir-33760f7d827
- https://ayyucekizrak.medium.com/deri%CC%87ne-daha-deri%CC%87ne-evri%C5%9Fimli-sinir-a%C4%9Flar%C4%B1-2813a2c8b2a9
- https://ayyucekizrak.medium.com/derin-%C3%B6%C4%9Frenme-ile-artistik-stil-transferi-29256789c7e8.
- https://colab.research.google.com/
- https://docs.python.org/3/library/socket.html
- https://docs.python.org/3/
- https://developer.android.com/reference/java/net/Socket
- https://developer.android.com/docs
- https://docs.aws.amazon.com/ec2/?id=docs_gateway
- https://numpy.org/doc/
- https://tr.wikipedia.org/wiki/CUDA
- https://developer.nvidia.com/Cuda-downloads
- https://docs.python.org/3/library/threading.html
- https://docs.python.org/3/library/multiprocessing.html

ŞEKİLLER LİSTESİ