FİGES

0/10/2023

Deep Learning
with MATLAB

Doğukan Mehmet
KILINÇ

FİGES

...../...../20.....

Image Datastores

ImageDatastores, bir veri kayrağına başvuran bir MATLAB değişkenidir. Görüntü sınıflandırma için derin öğrenme gerçekleştirilirken, görüntülerinizi bir imageDatastore'da saklamanıza yardımcı olur. Burada dosya adı, formatı, sınıflandırma etiketi gibi bilgiler saklanır. Ayrıca bu formata çevirilmiş bir veri deposu ile görüntülerin tamamını önceden kolay bir şekilde işleyebilirsiniz.

(1) `imageDatastore (Location)`

Tüm görselleri bir arada kullanmak için veya görüntülemek için "montage" kullanılır.

(2) `montage(datastore)`

Önceden eğitilmiş ağ atanması için aşağıdakini yapıyoruz.

(3) `net = googlenet;`

imageDatastore formatına çevirilen bir format içerisinde, içerisindeki tüm görseller için basit ön işleme gerçekleştirilebilir. Bu yeni veri deposunu (imageDatastore) görüntü giriş boyutunu giriş görüntülerine uygulamak için;

(4) `auds = augmentedImageDatastore([r c], imds)`

"classify" işleci bir görüntünün etiketlerini tahmin etmek için bir ağ kullanacaktır.

(5) `label = classify(net, ds)`

Transfer Learning

Transfer öğrenimi, yeni bir görüntü kümesini sınıflandırmak için önceden eğitilmiş bir ağın yeniden eğitilmesi sürecidir. Şunlar gereklidir;

- Network Layers (layers)
- Training Data (data)
- Algorithm Options (options)

Örneğin bir verisetindeki görüntülerin alt klasörlerde saklandığını varsayalım. Her klasörün adı o klasördeki görüntüler için kolaylık gelen etikettir. Veri deposundaki tüm görüntüleri bulmak ve etiketlenmek için "IncludeSubfolders" ve "foldernames" kullanılır.

(6)
```
ds = imageDatastore('location', ---
        "IncludeSubfolders", true, ---
        "LabelSource", "foldernames")
```

Eğitim sırasında ağ, eğitim görüntülerini ve etiketleri ilişkilendirmeyi öğrenir. Eğitim sırasında veriyi train ve test diye ikiye bölebiliriz.

(7)
```
[ds1, ds2] = splitEachLabel(imds, p)
```

p - oranı (0-1) arası değer alır ve belirlenen oranda bölme yapıp ds1'e kaydeder. Kalan ds2'ye kaydolur

Algoritma seçenekleri bir ağın nasıl eğitildiğini kontrol eder. Bu seçenekler "trainingOptions" fonksiyonu ile oluşturulur

(8)
```
opts = trainingOptions(algorithm, ---
        "InitialLearnRate", rate)
```

İlk girdi algoritma adıdır. Olası bir çok seçenek vardır. Bunu dilediğiniz gibi kontrol edebilirsiz.
Oluşturuba, elde edilen parametreler ile bir sistemi aşağıdaki kod ile train edebilirsiz.

⑨ `net = trainNetwork(trainds, lgraph, options)`

Ağı Değerlendirme

Bu değerlendirme confusion matrisi (karışıklık matrisi) ile olabilir.
Bir ağın yüksek eğitim ve test doğruluğuna sahip olması önemlidir. Test setindeki doğruluğu hesaplayarak ve karışıklık matrisini görüntüleyerek bir sınıflandırma ağının etkinliğini değerlendirebilirsiniz.
Yeni görüntülerdeki etiketleri tahmin etmek için aşağıdaki kod kullanılır.

⑩ `predictedLabels = classify(net, imds)`

Test görüntüsü veri deposundaki etiketleri ağdan tahmin edilen etiketlerle karşılaştırarak test doğruluğunu hesaplayabilirsiniz.

⑪ `nnz(trueLabels == predictedLabels)/n`

Sıfır olmayan ögelerin sayısını veya gerçek etiketin tahmin edilen etikete eşit olduğu ögelerin sayısını sayar.

Test görüntülerinin sayısıdır.

Test görüntülerinin sayısını bulmak için `length(testPred)`

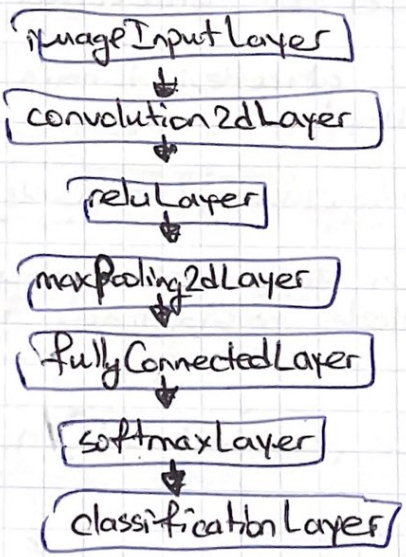
Karmalık matrisi (confusion matrix), hangi etiketlerin yanlış sınıflandırıldığını gösterir.

(2) confusionchart(known, predicted)

Burada x-ekseri ; tahmin edilen etiketi
y-ekseri ; bilinen etiketi gösterir.

AĞ MİMARİSİNİ ANLAMA

CNN mimarileri derinlik açısından çeşitlilik gösterir. Tüm CNN'ler görüntü giriş katmanı ve konvolüsyon katmanları gibi bazı ortak katmanlar içerir.



Yukarıdaki katmanlar basit bir derin ağ mimarisi oluşturur. Ağdaki her katman, girdileri üzerine bazı işlemler gerçekleştirir ve yeni bir değer çıktısı verir.

İlk katman görüntünün giriş katmanıdır. Bu katman ağın giriş boyutlarını tanımlar. ve görüntüleri normalleştirir.

Varsayılan olarak, bir görüntü giriş katmanı eğitim veri setinin ortalama görüntüsünü çıkarır.

Input Layer

convolution2dLayer, katmanları giriş görüntüsüne kayan filtreler uygular. Bu katman CNN mimarisinde önemli bir katmandır, katmanlardır-

reluLayer, konvolüsyon katmanlarını genellikle düzeltilmiş doğrusal birim (Relu) gibi doğrusal olmayan bir aktivasyon katmanı takip eder. Bir Relu katmanı, girdinin her bir öğesi için bir eşik işlemi gerçekleştirir. Sıfırdan küçük herhangi değeri sıfıra eşitle.

maxPoolingLayer, bir maximum havuzlama katmanı, girişi dikdörtgen havuzlama bölgelerine bölerek ve her bölgenin maximumunu hesaplayarak aşağı örnekleme gerçekleştirir.

Havuzlama, ağ karmaşıklığını azaltır ve daha genel bir ağ oluşturur.

fullyConnectedLayer, ağdan geçen özellikler, tam bağlantı katmanına ulaşıncaya kadar bir matris formatında saklanır. Tam bağlantı katmanında girdi, çıktı sınıfı ile etkileştirilebilmesi için "düzleştirilir". Bu katman klasik bir sinir ağıdır. Bu katman için çıktı boyutu, sınıflandırma problemimiz için sınıf sayısıdır.

softmaxLayer, normalleştirilmiş üstel fonksiyon kullanarak her çıktı sınıfının değerini normalleştirilmiş puanlara dönüştürür. Her değeri, giriş görüntüsünün her sınıfa ait olma olasılığı olarak yorumlayabiliriz.

classificationLayer, sınıflandırma çıktı katmanı en olası sınıfın adını döndürür

CNN'lerdeki konvolüsyon katmanları, öğrenilen filtreleri kullanarak konvolüsyon gerçekleştirir.

Konvolüsyon Oluşturma

Bir RGB görüntünün 3 farklı kanalı vardır. Konvolüsyon gerçekleştirmek için de iki boyutlu bir matrise ihtiyacımız vardır. İki nokta üst üste (:) operatörünü kullanarak kanallardan birinin tüm satır ve sütunlarını çekebiliriz.

(13)
```
image (:, :, n)
```

Daha önceden oluşturulmuş filtre matrisini görüntüye uygulamak için aşağıdaki kodu kullanırız.

(14)
```
filteredim = conv2 (filtre, im)
```

Gri tonlamalı görüntüleri görüntüleri ekrana bastırırken imshow'un yanına gerekli ikinci bir argüman eklemek gerekir.
İkinci girdi olarak boş parantez kullanmak, görüntüde bulunan minimum ve maksimum değerlere göre ekranı ölçekleyecektir.

(15)
```
imshow (im, [])
```

Farklı filtreler de uygulayabiliriz.

Aktivasyon fonksiyonu ile bir CNN kullanarak bir giriş görüntüsünden özellikler çıkarabiliriz. Bu fonksiyon 3 girdi kabul eder. net, görsel ve özelliklerin çıkarılacağı katman.

(16)
```
features = activations (net, img, layer#)
```

Bazen ~~değişkenler~~ ilgilenilen layer'da, katmanda birden fazla kanal vardır. Bu nedenle tüm kanalları ayrı ayrı incelemek zordur.
"showActivationsForChannel" fonksiyonu ile birlikte belirli bir kanaldaki aktivasyonlar görüntülenir.
Beyaz bir piksel, kanalın o konumda pozitif olarak etkinleştirildiğini, siyah piksel ise kanalın negatif olarak etkinleştirildiği yerdir.

(17) `showActivationsForChannel(img, feature, nth)`

Konvolüsyon katmanının hiperparametreleri ve öğrenilebilir parametreleri vardır. Hiperparametreler ağ oluşturulduğunda ayarlanır ve eğitim sırasında değişmez. Öğrenilebilir parametreler eğitim sırasında güncellenir.
Bu değerler Weights'ta saklanır. Örneğin:

```
Convolution2DLayer with properties:

    Name: ''
Hyperparameters
    FilterSize: [7 7]
        :
Learnable Parameters
    Weights: [7×7×3×64 single]
    Bias: [1×1×64 single]
```

Bu evrişimli katmanın ağırlıkları dört boyutlu bir dizide saklanır. Dördüncü boyut (64) kanal sayısı

Bir katmandaki tüm kanalları görmek için önce aktivasyon almamız gerekir.

(18) `actv = activation(net, img, Layer)`

Aktivasyonlar herhangi bir değer alabilir bu nedenle aktivasyonları görüntülemeden önce ölçeklendirmek faydalı olacaktır. Bunu "mat2gray" ile yapabilirsiniz

(19) `actvnGray = mat2gray(actvn)`

Derin öğrenme ağını eğitmek için en iyi yaklaşım TRANSFER LEARNING'dir. (Görseller RGB ve aynı boyutta değil)

Görseller Gray-scale ve aynı boyuttaysa en iyi öğrenme yaklaşımı bir AĞI SIFIRDAN EĞİTMEK (Train a Network from Scratch) tır

Gri tonlamalı görüntülerde ⇒ a x b x 1
Renkli (RGB) görüntülerde ⇒ a x b x 3 kanal vardır.
Aroni Görüntüsü Görüntülerinde ⇒ a x b x 4 kanal vardır.
↓
Kırmızı
Yeşil
Mavi
Yakın Kızılötesi

Evrişim katmanları, görüntüye farklı filtreler uygulayarak giriş görüntüsündeki özellikleri öğrenir Evrişim katmanını (convolution layer) oluşturmak için filtre boyutunu ve filtre sayısını belirtmemiz gerekir

(20) `convolution2dLayer([h w], n)`

↳ FilterSize → [h w] → h x w boyutunda,
↳ NumFilters → n → n tane filtreli bir convolution layer.

Evrişim katmanlarını (convolution layer) genellikle düzeltilmiş doğrusal birim (ReLu) ve maximum havuzlama katmanı tahmin eder.

ReLu katmanı tüm negatif değerleri sıfıra ayarlar.

(21) reluLayer()

Maksimum havuzlama katmanları, dikdörtgen bölgeleri bir araya toplayarak ve her bölgenin maksimumunu hesaplayarak alt örneklendirme gerçekleştirir. Havuzlama boyutu giriş olarak kullanılır.

(22) maxPooling2dLayer([h w])

Evrişimsel sinir katmanlarının son üç katmanı aşağıdaki gibidir.

- fullyConnectedLayer
- softmaxLayer
- classificationLayer

Fully-Connected-layer, ağın tahmin edebileceği sınıf sayısını temsil eden Outputsize özelliğine sahiptir.

FİGES

16./10./2023

Ufak Bir Örnek

trainingOptions ile yapılan eğitimi grafik üzerinden incelemek istersek aşağıdaki kodu yazarız

(23)
```
trainingOptions ("plots", "training-progress")
```

Ağı eğitmek için hazırız. Eğitim görüntülerini ve etiketleri XTrain ve YTrain olmak üzere iki farklı değişkene kaydedilir. Kod aşağıdaki gibidir

(24)
```
net = trainNetwork(X, Y, Layers, opts)
```

Test verileri bir XTest değişkeninde saklanır ve ilgili etiketler YTest'te bulunur.
Ağda tahminlerde bulunmak için "classify" kullanabiliriz.

(25)
```
testpreds = classify (net, Xtest)
```

Ardından karışıklık tablosunu görmek için (confusion matrix) aşağıdaki kodu kullanırız

(26)
```
confusionchart (known, predicted)
```
known: Bilinen, predicted: Tahmin Edilen

Şimdiye kadar bir ağı eğitimden sonra bir test veriseti kullanarak değerlendirdik. Ayrıca bir doğrulama veriseti de kullanabiliriz. Doğrulama veriseti, eğitim sırasında ağı değerlendirmek için kullanılan verilerin ayrı bir bölümüdür

Training Data: Eğitim, Öğrenme sırasında ağırlıkları güncellemek için kullanılır.

Validation Data: Performansı değerlendirmek için eğitim sırasında kullanılır.

Testing Data: Performansı değerlendirmek için eğitimden sonra kullanılır.

Doğrulama verileri, ağın aşırı uyum sağlayıp sağlamadığını tespit etmek için kullanışlıdır. Eğitim kaybı azalsa bile doğrulama kaybı artıyorsa eğitim durdurulmalıdır. Çünkü ağ, eğitim verileri hakkında yeni görüntülerle ilgili olmayan ayrıntıları öğrenmektedir.

(27) `[trainds, valds, testds] = splitEachLabel(imds, 0.8, 0.1)`

- trainds → data
- valds → Validation data
- testds → Test data
- 0.8 → %80 train
- 0.1 → %10 validation

Doğrulama verileri bir veri deposu, tablo veya hücre dizisi olabilir. Bu veri seti için görüntüler ve etiketler iki farklı değişkende saklandığından bir hücre dizisi oluşturmak basittir.

Doğrulama verilerinden oluşan bir hücre dizisinin iki öğesi olmalıdır; bunlardan ilki görüntüleri, ikincisi ise etiketleri içerir.

(28) `validationData = {X, Y}`

- X → Görüntüler
- Y → Etiketler (Hiçbir şey yok!)

Doğrulamayı da görmek için "ValidationData" seçeneğini ekleyebilirsiniz.

Ağ çok sık bir şekilde eğitildiğinden, ağın daha sık doğrulaması için doğrulama sıklığı da azaltılabilir.

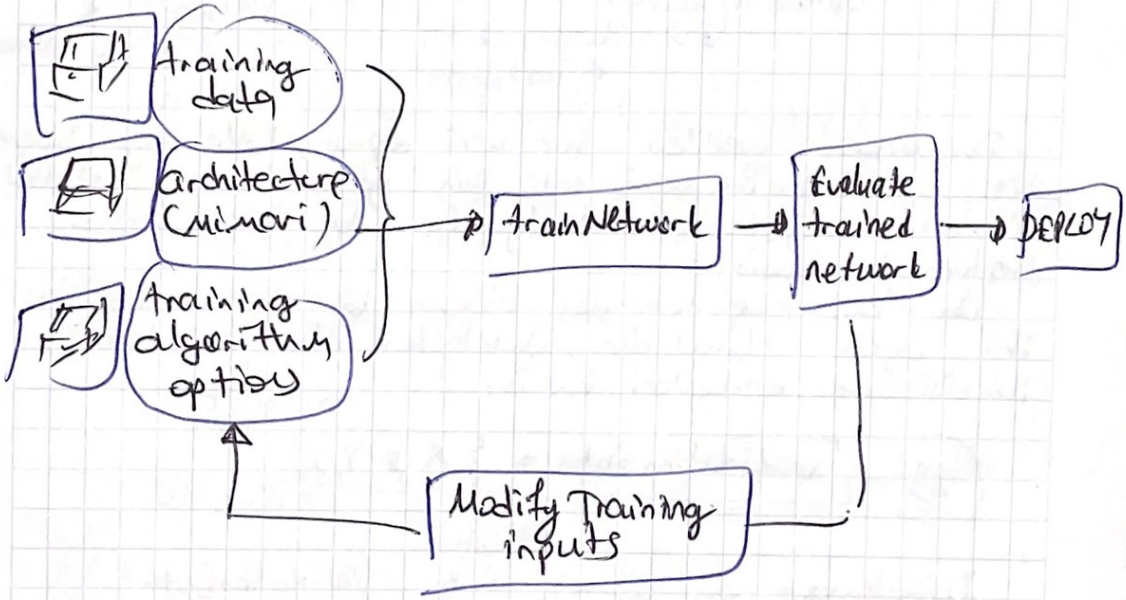
(29)
```
trainingOptions("ValidationData", data, ...
    "ValidationFrequency", freq)
```

Validation doğruluğu, eğitim doğruluğuna benzer olduğuna dikkat edilmeli. Bu ağın aşırı uyum sağlamadığına işarettir ve iyi bir durumdur

AĞ PERFORMANSI İYİLEŞTİRMEYE GENEL BAKIŞ

Derin öğrenme için 3 şeye ihtiyaç vardır.
1) Bilinen etiketlere sahip veriler (Data)
2) Ağ mimarisini temsil eden katmanlar (Labels)
3) Eğitim algoritması parametreleri (options)

Ağ eğitildikten sonra bir test verisi ile doğruluk hesaplanır. Performanstan memnunsak bu ağı üretimde kullanmaya başlanır.



Bazı sınıflandırma sonuçlarımızda istenilen sonucu vermeyebilir. Bunun sebebi ilgilenilen bölgenin istenilen bölge olmaması olabilir. İlgilendiği (sistemin) bölgeyi görmek için Occlusion hassasiyeti adlı tekniği kullanabiliriz.

Occlusion hassasiyet haritasını hesaplamak için 3 girdiye ihtiyaç var.

(32) `m = occlusionSensitivity(net, I, label)`

↳ pred, classify sonucu oluşan çıktı

Bu haritayı görüntülemek için ilgili resim üzerine oturtmamız gerekmektedir. Öncelikle "imshow" ile ilgili resimi görüntüleriz.

(31) `imshow(img)`

Ardından hassasiyet haritasını (occlusionSensitivity) kaplamak için "imagesc" fonksiyonunu düşük bir alfa değeri ile kullanabiliriz.

(32) `imagesc(I, "AlphaData", a)`

↳ 0-1 arasıdır. 0; saydam, 1; opaktır.

Yani örnek kod aşağıdaki (img) gibidir.

(33)
```
map = occlusionSensitivity(net, I, label)
imshow(img)
hold on
imagesc(map, "AlphaData", a)
hold off
```

FİGES

...../...../20.....

Aşağıdaki kod ile hassasiyet haritasını görsel üzerinde görebiliriz. Ayrıca "colorbar" ile bir renk çubuğu da ekleyebiliriz.

(34)
```
colormap jet
colorbar
```

Bu haritada kırmızı ile görülen yerler classify'de tahmin edilen içeriklerdir. Fakat zaman tahmin edilen yer ilgilendiğimiz alan olmayabilir. Bunun için hassasiyeti değiştirebiliriz.

(35)
```
catmap = occusionSensitivity(net, img, "tabby")
imshow(img)
hold on
imagesc(catmap, "AlphaData", 0.5)
hold off
colormap jet
colorbar.
```

Afg arka tarafta tahmin yaptığı değerleri saklıyordur. Bu değerler arasından "tabby"nin nereye denk geldiğini ısı haritasında görmüş olduk.

FİGES

...../...../20.....

Image Augmentation

Görüntüleri döndürme, yansıtma, kesme, öteleme kombinasyonlarıyla dönüştürmek için "augmentedImageDatastore" kullanabiliriz.

Ağ, eğitim verilerinin tam ayrıntılarını ezberlemediği için veri artırma, aşırı uyumu önler.

Ağı eğittiğimizde, veri deposundan sağladığımız ayarları kullanarak görüntüleri rastgele artıracaktır. Dönüştürülen görüntüler bellekte saklanmaz, bu da büyük veri setleri üzerinde eğitim yaparken kullanışlıdır.

"imageDataAugmenter" işlevi, büyütmeyi seçmek için kullanılabilir. Olası büyütmeler yansıma, öteleme ve ölçeklendirme gibi dönüşümler içerir.

Verisetini büyütmek için görüntüleri rastgele bir miktarda döndürelim.

(36) `imageDataAugmenter("RandRotation", [min max])` → augmenter

Bir artırılmış görüntü veri deposu oluştururken, çıktı görüntü boyutunun, dosyaların kaynağını ve augmenter'ı (bir üst koddaki) belirtmeliyiz.

(37) `augmentedImageDatastore(size, ds, "DataAugmentation", augmenter)` → augImds

"read" fonksiyonu ile tüm verileri okuyabilir, boyutlarını görebiliriz.

(38) `data = read(augImds)`

Yukarıdaki data'dan görüntü açmak ve ekrana yazdırmak için:

(39) `imshow(data.input{n})`

FİGES

17./12./2023

Hiperparametre Optimizasyonu

Olası eğitim parametrelerini keşfetmemiz gerektiğinde MATLAB'da bir deney oluşturabiliriz. "Experiment Manager" uygulaması, ağların verilerimiz üzerinde uygun şekilde eğitildiği ve sonuçların karşılaştırma için gösterildiği, değişken hiperparametrelerle bir dizi deneme oluşturmamıza olanak tanır. Bunu Toolslar arasındaki Experiment Manager üzerinden gerçekleştirebiliriz.

Regresyon Nedir?

Konvolüsyonel sinir ağları genellikle görüntü sınıflandırması için kullanılır.

Daha genel olarak sınıflandırma, yeni veriler için ayrı bir yanıt öngören bir tahmin modeli oluşturmak için verilen ve bilinen yanıtları kullanma görevini ifade eder. Görüntü sınıflandırması için girdi verisi görüntüdür ve bilinen yanıt görüntü örneğinin etiketidir.

REGRESYON:

Derin öğrenme ile gerçekleştirilebilecek bir başka görevdir Regresyon, verilere ayrık sınıflar yerine sürekli yanıt değerleri atamak anlamına gelir.

Regresyon için Veri Hazırlama

Regresyon için transfer öğrenimini gerçekleştirmek üzere "trainNetwork" fonksiyonunu verilerimizi, mimarimizi ve eğitim seçeneklerini girdi olarak kullanabiliriz.

Etiketler kategorik olmadığı için regresyon gerçekleştirirken eğitim verilerimizi depolamak için imageDatastore kullanamayız. Uygun bir alternatif tablo veri dizisi kullanmaktadır.

Tablonun ilk değişkeni her bir değişkenin dosya dizinini ve adlarını içermektedir. İkinci ve sonrası yanıt elemanlarıdır.

Bu görüntüleri görüntülemek için öncelikle tablo değişkeni dosya adlarını veriden çekmek gerekir. Bunun için nokta gösterimi kullanılır.

(42) `tableName.VariableName` ≠ trainfiles

Hücre dizisinden veri çekmek için köşeli parantez kullanırız.

(41) `fn = trainfiles{n}`

↳ Ardından bunu "imshow" ile görüntüleyebiliriz.

Eğer elimizde imageDatastore oluşturacak veriler çoksa ve biz ilk 9 görüntüyü alıp 3x3 matriste hepsini yerleştirmek istersek

(42) `imds = imageDatastore(trainfiles(1:9)); montage(imds)`

Bu ilk 9 veriyi ve Color değerlerini görmek için;

(43) `tableName(rows, cols)`

RENK İÇİN ÖRNEK ↳ `subsetData = trainingData(1:9, :)`

↓ ÇIKTIDAKİ

	File	Color 1	2	3
1. satır	D:/...	5	-10	-2
		↓ Red	↓ Green	↓ Blue

Kırmızı baskın renk demek

Toplu boyutlandırma için

(44) `auds = augmentedImageDatastore([r c], table)`

- Regresyon problemlerinde "SoftmaxLayer" ve "classification" katmanlarını kaldırırız.
- Bunun yerine "RegressionLayer" katmanını koyarız.

Bir regresyon ağı eğitirken ⇒ RMSE, Loss değerlerine bakılır.

augmentedImageDatastore ⇒ Görüntüleri toplu boyutlandırır.

predict (net, ds) ⇒ Tahmin işlevi yapılır.

TestData.Color ⇒ TestData içerisinden Color'u çeker.

Tahmin edilen RGB değerinin kök-ortalama-kare-hatası (RMSE) hesaplanacak. Bu yöntem regresyon ağının doğruluğunu hesaplamak için yaygın bir yöntemdir.

(45) `error = rmse(pred, actual)`

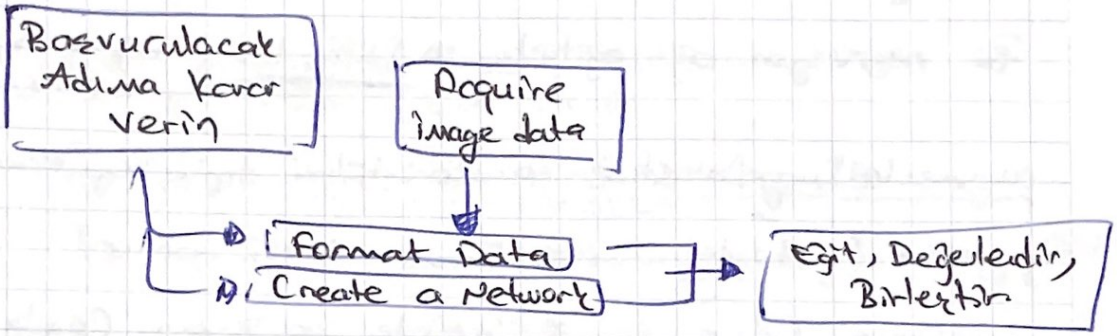
- pred: Tahmin edilen değerler (predict'ten)
- actual: Test veri setinden elde edilen değerler (TestData.Color'dan)

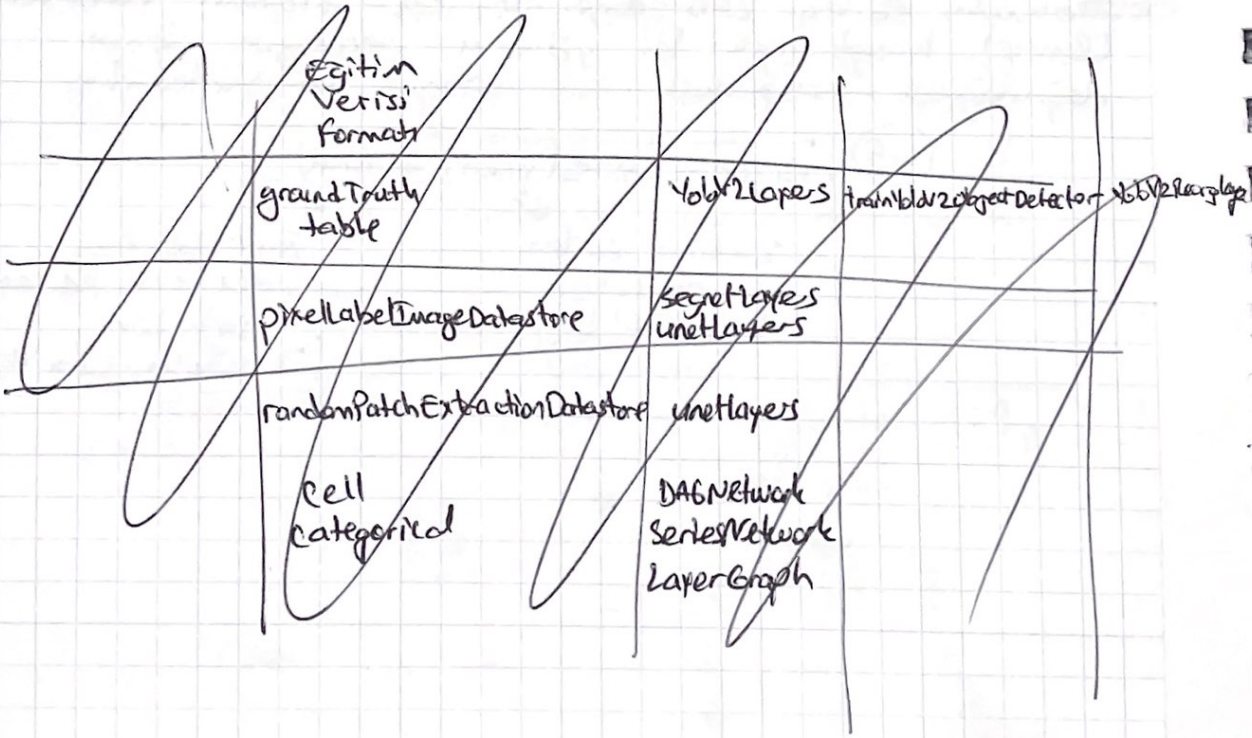
FİGES

(Bilgisayarlı Görme) ...../...../20.....

Computer Vision Uygulamalarının Karşılaştırılması

Derin öğrenme teknikleri çeşitli uygulamalara uyacak şekilde görüntülere farklı şekillerde uygulanabilir. Hedefinize ve verileri nasıl etiketlediğinize bağlı olarak, aşağıda listelenen birçok örnek için aynı veri setini kullanacağız.





	Eğitim Verisi Formatı	Network Mimarisi	Eğitim ve Test Fonksiyonları	Uygulama için katman Tipleri
Nesne Algılama (Object Detection)	groundTruth table	YoloV2Layers	trainYOLOv2ObjectDetector detect	yolov2ReorgLayer yolov2TransformLayer yolov2OutputLayer
Anlamsal Segmentasyon (Semantic Segmentation)	pixelLabelImageDatastore	segnetLayers unetLayers	trainNetwork semanticseg	image3dInputLayer convolution3dLayer pixelClassificationLayer
Görüntüden Görüntüye Regresyon	randomPatchExtractionDatastore	unetLayers	trainNetwork predict	regressionLayer transposedConv2dLayer
Video Sınıflandırma	cell categorical	DAGNetwork SeriesNetwork LayerGraph	trainNetwork classify	sequenceInputLayer bilstmLayer sequenceFoldingLayer sequenceUnfoldingLayer

FİGES

www.figes.com.tr

...../...../20.....

Önceden eğitilmiş bir Ground Truth'un içerisindeki dosyalara erişmek için aşağıdaki kodu yazarız.

(46)
```
im = imread(petGroundTruth.imageFilename{n});
```
n. görsel

Çoğu satır yalnızca bir sınırlayıcı kutu içerir. Ancak bazen (1 x n double) formatında veriler de içerebilir. Bunları görmek için aşağıdaki kodu kullanmalıyız.

(47)
```
b = groundTruth.Label(1)
ÇIKTI ↳ {1 x 4 double}

b = groundTruth.Label{1}
ÇIKTI ↳ 65 100 150 78
```
→ KÖŞELİ PARANTEZ ÖNEMİ 0

Görüntüyü etiketiyle birlikte görüntülemek için görüntüye sınırlayıcı kutu ekleyebiliriz.

(48)
```
alteredImg = insertObjectAnnotation(image, ---
    "rectangle", bbox, label)
```
- Dikdörtgen etiket
- Etiketlenecek koordinat
- Görsel
- Etiketlenen alanın işaretlendirilmesi

(49)
```
imshow(alteredImg)
```
ile ekrana yazdırız

Bir çok nesne dedektörü görüntülerin aynı boyutta olmasını gerektirmez ancak eğitim verilerinizi tutarlı bir boyuta ölçeklendirmek, eğitim sırasında sorunların önlenmesine yardımcı olabilir. Sınırlayıcı kutular piksel konumlarıyla birlikte saklandığından, görüntüler ölçeklenince sınırlı kutuların da ölçeklendirilmesi gerekir.

Öncelikle imageDatastore oluşturucuz.

(50) `imds = imageDatastore (petGT.imageFilename)`
↳ Dosya yolları alınarak imageDatastore oluşturulur

BoxLabelDatastore fonksiyonu, sınırlayıcı kutu etiketi verileri için bir veri deposu oluşturur.

(51) `ds = boxLabelDatastore (tbl)`
↳ Sınırlayıcı kutuları içeren bir tablodur. Her sütun bir nesne sınıfını temsil eder.

En başta dosya uzantıları dan table dosyasında 2'den sona kadar etiketleri boxLabelDatastore ile alacağız.

(52) `bxds = boxLabelDatastore (table (:, 2:end))`

2'den en sona dan tüm sütunları alıncı

(51) ve (52)'de görüntüler ve sınırlayıcılar saklanmaktadır. Birlikte işlemek için bunları ~~işlemek~~ birleştirmemiz gerekir

(53) `dsnew = combine (ds , bxds)`

ds1 → veri deposu görüntü

ds2 → kutu etiketi veri deposu

Eğitim görüntülerin boyutları farklı olabilir. Görüntüleri yeniden boyutlandırırsak, ilgili sınırlayıcı kutuları da yeniden boyutlandırmak gerekir. "transform" fonksiyonu mevcut veri depolarını önceden işleyebilir.

(54)
```
dataout = transform(datain, @myfun)
```
fonksiyonu bunda "@" ile beraber kullanırız.

Örnek bir fonksiyon

Ön işleme için bir fonksiyon tanıtıcısı.

Bu kodda scaleGT görüntüleri targetSize olarak yeniden boyutlandırılır. Aynı ölçek ilgili sınırlayıcı kutular yeniden boyutlandırmak için de kullanılır.

(55)
```
function data = scaleGT(data)
    targetSize = [224 224];
    % data{1} is the image
    scale = targetSize./size(data{1},[1 2]);
    data{1} = imresize(data{1}, targetSize);
    % data{2} is the bounding box
    data{2} = bboxresize(data{2}, scale);
end
```

(54) sonucu oluşan görüntüyü izlemek için "preview" kullanılır.

(56)
```
p = preview(ds)
```

Sonuç olarak 2'li elemanlı bir hücre dizisi oluşur.

1) Scaled image (Ölçeklendirilmiş Görüntü)
2) Scaled bounding box (Ölçeklendirilmiş sınırlayıcı kutu)
3) Label (Etiket)

Sonucu görüntülemek için ölçeklendirilmiş görüntüye açıklama ekleyebiliriz.

```
57  alternedImg = insertObjectAnnotation(image, ...
                  "rectangle", bbox, label)
```

Burada da

```
    im = insertObjectAnnotation(p{1}, ...
         "rectangle", p{2}, p{3})
```

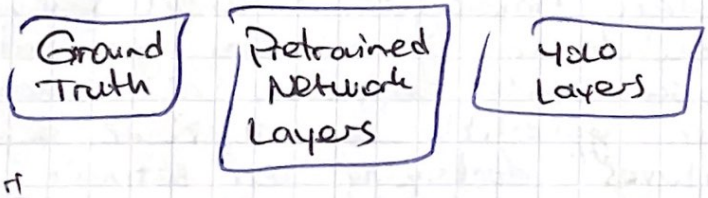
kullanırız

YOLO v2 MİMARİSİ

You-only-look-once (YOLO) ağları, her görüntüye yalnızca bir kez bakan, tek aşamalı nesne tespiti gerçekleştirmek için kullanılır. Bir YOLO ağı, tespit edilen nesnelerin ölçeğini ve en-boy oranını karakterize eden bağlantı kutularına dayanır.

Bir YOLO v2 nesne algılama ağı iki alt ağdan oluşur;

1) Özellik Çıkarma Ağı
2) Algılama Ağı



Önceden eğitilmiş bir ağ ve bağlantı kutuları verilen bir YOLO v2 nesne algılama ağı düzenlemek için "yolov2Layers" işlevini kullanırız.

```
58  lgraph = yolov2Layers(imageSize, numClasses, anchorBoxes, ...
                net, featureLayer, "ReorgLayerSource", reorgLayer)
```

OUTPUTS:
- layerGraph: Birleştirilmiş YOLO v2 nesne algılama ağı

INPUTS:
- imageSize: Ağ için giriş görüntüsü boyutu
- numClasses: Veri setindeki nesne sınıfı sayısı
- anchorBoxes:
- net: Önceden eğitilmiş konvolüsyonel sinir ağı
- featureLayer: Özellik çıkarma için kullanılacak ağdaki daha derin katmanlardan birinin adı (tipik olarak bir ReLu katmanı)
- "ReorgLayerSource": YOLO katmanının ağdaki farklı derinliklerden gelen özellikleri dikkate alınmasını sağlar. (İSTEĞE BAĞLI GİRDİDİR.)
- reorgLayer: Ağda featureLayer'dan önceki bir katmanın adı. (Genellikle başka bir ReLu katmanı)

Bir nesne dedektörünü eğitmek için bağlantı kutularına ve bir ağ mimarisine de ihtiyacımız olacaktır. Bağlantı kutuları, veri setindeki nesnelerin ölçeğine ve en-boy oranını karakterize ederek nesne dedektörünün performansını arttırır.

YOLO dedektörleri sınırlayıcı kutuları bulmak için bağlantı kutularına güvenir. Sınırlayıcı kutuların boyutları biliniyorsa bunlar doğrudan ağ mimarisine aktarılabilir. Ancak genellikle durum böyle değildir. "estimateAnchorBoxes" fonksiyonu veri setimiz için uygun bağlantı kutularını bulacaktır.

(59) `bxs = estimateAnchorBoxes(data, numBxs)`

Bağlantı kutu sayısı (5) Beşli

ÖRNEK:

```
net = resnet18;
numClasses = 5;
imageSize = [224 224 3];
```

Artık veri setine uygun bağlantı kutularına sahip olduğumuza göre, YOLO mimarisi için kodlar girdiler oluşturulabilir.
YOLO v2 mimarisi aşağıdaki gibi eğitilir.

(60)
```
layers = yolov2Layers(sz, n, box, net, ...
                      featureLayer, "ReorgLayerSource", reorgLayer)
```

sz: imageSize
n) numClasses
box: anchorBoxes
net: net
featureLayer: Özellik çıkarım katmanı
reorgLayer: featureLayer'den önceki katmanın adı.

BURAYA KADAR LAYERGRAPH kısmı oluşur.

(61) BİR YOLO DEDEKTÖRÜNÜN EĞİTİMİ

```
detector = trainYOLOv2ObjectDetector(trainingData, layerGraph, options)
```

OUTPUT: detector: Eğitilmiş YOLO ağı
INPUT: trainingData: Eğitim verisi katmanları
layerGraph: YOLO v2 içeren ağ mimarisi
options: Eğitim ayarları

Birden fazla görüntüyü test etmek istiyorsak sayısal bir metrik hesaplamak faydalı olabilir.

Hassasiyet Hesaplama:
Önceki sınıflandırma modellerinde, doğruluğu yanlızca yanlız sınıflandırmaya göre hesaplıyorduk. Artık hem sınırlayıcı kutuya hemde etiketi dikkate almalıyız.
"evaluateDetectionPrecision" fonksiyonu, tahmin edilen ve gerçek sınırlayıcı kutular arasında bir örtüşme eşiği kullanarak bir keskinlik ölçütü hesaplar.

Ortalama değerler (PRECISION) = TRUE POSITIVE / ALL POSITIVE

FİGES

...../...../20.....

fx evaluateDetectionPrecision

Kayıp Oranı

Dedektörün bir nesneyi bulmadığı durumları da göz önünde bulundurmalıyız. Buna miss rate denir. "evaluateDetectionMissRate" fonksiyonu kullanılır.

fx evaluateDetectionMissRate

Veri depoları, birden fazla dosyada depolanan verilere erişmek için uygun bir yol sağlar. Sinyal verileri için bir sinyal veri deposu oluşturabiliriz.

(62)
```
ds = signalDatastore("folder", ---
     "IncludeSubfolders", true)
```
→ Verilen klasörün alt klasöründeki simgeleri dahil için

Verileri veri deposundaki bir dosyadan almak için "read" fonksiyonu kullanabiliriz.

```
data = read(ds)
```
→ Datastore verisi girilir.

Okuma fonksiyonu ilk kez kullanıldığında veriler ilk dosyadan içeri aktarılır. İkinci kez kullanıldığında veriler ikinci dosyadan içeri aktarılır ve böyle devam eden

(63) (Koordinatları eklemek için)
```
ds = signalDatastore("folder", "SignalVariableNames", ["Var1" "Var2"],
     "ReadOutputOrientation", "row"
                              "columns"
```
→ Bunları kullanarak satırları veya sütunları birleştirebiliriz. Tek satır matris edebiliriz

data = read(ds) ile okuduğumuz dosyanın formatı bir hücre dizisidir. Sinyali görselleştirmek için "cell2mat" ile bir matrise dönüştürebiliriz.

(64) `datamatrix = cell2mat(datacell)`

Bu işlemden sonra `plot(datamatrix)` ile görselleştirebiliriz. Eğer;

(65)
```
varnames = ["ax" "ay" "az"]
legend(varnames)
```

yazarsak hem grafiği çizecek hem de çizgilerin hangi renginin ne olduğunu gösterecek.

Veri setine erişmek için signalDatastore kullanırız. Sinyali sınıflandırabilecek bir ağı eğitmek için verilerimizin etiketlenmesi gerekir. Bu etiketlerin konumu verisetine bağlı olacaktır.

Her bir sinyalin etiketlerini almak için "folders2labels" fonksiyonu kullanılabilir.

(66) `lbs = folders2labels("folde")`

lbs değişkeni kategorik bir vektör içerir. Veri setinizin hangi etiketlere sahip olduğunu ve sınıf başına düşen sinyal sayısını görmek için kategorik verilerle özet işlevini kullanabiliriz.

(67)
```
lbs = folders2labels("folde")
summary(lbs)
```
→ Hangi kategoriden kaç tane veri var onu gösterir.

Ağı eğitmeden önce etiketlerde değişiklik yapmak isteyebiliriz. Örneğin iki farklı sınıfı tek bir sınıfta birleştirebiliriz.

(68) `lbls = renamecats(lbls,["class1" "class2"])`

LABEL KATEGORİLERİN ADINI DEĞİŞTİREBİLİRİZ.

Veri seti etiketlendiğine göre her birinden gelen sinyallere bakabiliriz. Her sınıftan bir sinyal bulmak için aşağıdaki kodunu kullanabiliriz.

(69) `[lb, idx] = unique(lbs)`

lb, sınıflarımızın adını → Etiket adları
idx, her sınıf için ilk indeksi verir. → İlk değerin kaçıncı indexte olduğunu verir.

Ardından bu ~~sinyallerdeki~~ indexlerdeki sinyalleri ayıklamak için sinyal veri deposundaki "subset" işlevini kullanabiliriz.

(70) `dsidx = subset(sigds, idx)`

Bir veri deposundaki tüm sinyalleri içe aktarmak için "readall" fonksiyonu kullanılır.

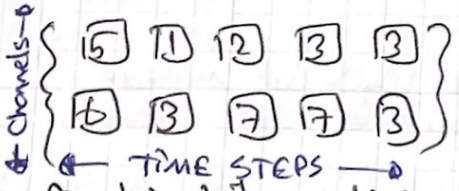
(71) `data = readall(ds)` ← dsidx

↳ Bunu kullanmamızın sebebi, sinyalleri görüntülemek için sinyalleri dosyalardan çalışma alanına aktarmamızın gerekliliğidir.

Veri depoları, verilerinizi yüklemeye başlamak için iyi bir yoldur çünkü derin öğrenme için veri setleri genelde çok büyüktür.

Bir sinyal veri deposu, sinyallerin hepsini belleğe yüklemeden erişmeyi sağlar. Derin ağları eğitmek için girdi olarak bir sinyal Datastore kullanabiliriz fakat bunun dezavantajı da olmaktadır. Bu da dosyaları bilgisayarımızdan MATLAB'a okutmanın yavaş olabilmesidir.

Veri Setinin Bir Hücre Dizisinde Saklanması



- Ağı bir hücre dizisi ile eğiteceksek tüm veri seti tek sütunlu bir hücre dizisinde saklanacaktır.
- Yukarıdaki hücre dizisindeki her bir öğe bir gözlem veya dizidir. Bu gözlemler sayısal matrisi oluşturur.
- Her gözlemdeki sütunlar zaman adımlarıdır. Her gözlem farklı sayıda zaman adımına sahip olabilir. Ancak sinyaller eğitim için aynı uzunluğa sahip olacak şekilde kesilecek veya doldurulacaktır. (TIME STEPS)
- Satırlar, gözlemlerin özellik boyutuna karşılık gelir. Veri setindeki farklı kanallar, farklı sensörlerden gelen sensör sinyalleridir. Tüm gözlemlerin aynı sayıda kanala sahip olması gerekir. (CHANNELS)

VERİ SETİNİ DÖNÜŞTÜRME

Önceliklе aşağıdaki fonksiyonu tanımlayalım:

(72)
```
function sig = prepsig(sigin)
    sig = cell2mat(sigin)';
end
```

Bu fonksiyon görselleştirme için dönüşüm yapar. Sinyal veri deposu bir dönüştürme fonksiyonu kullanarak dönüştürülebilir.

(73)
```
dst = transform(ds, @function)
```

@prepsig gibi yazılır.

signalDatastore

Bir fonksiyonun başka bir fonksiyona nasıl aktarılacağını gösteren bir fonksiyon tanıtıcısı.

Tüm veri setini ile aktarmak için → `readall(ds)` kullanılır.

"readall" sonrası tüm gözlemler büyük bir matrise aktarılır. Gözlemleri birbirinden ayırmak için işlemiz sinyali hücrede saklayabiliriz.

Bu duruma başvurmamızın sebebi bir LSTM eğitilirken, ağ bunu onbinlerce kanala sahip tek bir gözlem olarak düzenliyor olmasıdır.

(74) `sigout = {sigout}` → fonksiyonun sonuna ekleriz (prepsig)

Sigout'taki hücrelerin boyutları x,y,z yönündeki uzaya karşılık gelen üç sütunlu bir matris olabilir.

Kanallar sütunlarda saklanır ancak bu verileri, bir ağı eğitmek için kullanmak için SATIRLARDA saklanması gerekir. Bunu matrisin transpoz (') ederek düzeltebiliriz.

(75) `row = col'` → `sigout = sigout'` bunu function içinde (74)'ten önce kullanılır.

Eğitim sırasında ağ, eğitim sinyallerini ve etiketleri ilişkilendirmeyi öğrenir. Ağ, yüksek bir eğitim doğruluğuna sahip olabilir. Ancak ağ, yeni sinyallere genelleme yapabiliyorsa kullanışlıdır. Ağın hiç görmediği sinyalleri sınıflandırıp sınıflandıramayacağını değerlendirmek için ayrı bir doğrulama ve test veri seti kullanmalısınız. Veri setini "splitlabels" kullanarak bölebilirsiniz.

(76) `idx = splitlabels(labels, [0.9, 0.05])`

~~Tüm veri setinin etiketi~~ Eğitim verisi oranı

validation, doğrulama verisi oranı

Kalan %5'te test için otomatik atanır.

İndisler her bir elemanın sırasıyla eğitim, doğrulama ve test indislerini içerdiği bir hücre dizisinde saklanır. Örneğin eğitim indexlerini şu şekilde alabiliriz.

(77)
```
idx{1}
```

İndexleri çıkarttıktan sonra, bunları indexleme kullanarak veri kümesini bölümlemek için kullanabiliriz. data ve labels hizalanır çünkü fileDatastore labels işlevi dosyaları signalDatastore ile aynı hızda okur. Bunları görüntüleyemek için;

(78)
```
trainidx = idx{1}
validx = idx{2}
testidx = idx{3}
```

Eğitim verilerini ve ~~ordinary~~ etiketleri normal dizi indexleme ile çıkarabilirsiniz.

(79)
```
training = dataset(idx)
```

Eğitim verileri genellikle veri setinin en yüksek oranıdır çünkü ağı eğitmek için kullanılır.

Ağ, doğrulama verileri kullanarak eğitilmez, ancak ağın yeni sinyallere genelleme yapıp yapmadığını izlemek için eğitim sırasında doğrulama verileri kullanılır.

Eğitim tamamlandıktan sonra, test verileri nihai ağı test etmek için kullanır. Doğrulama ve eğitim için aynı gözlem kullanılabilir. Ancak ayrı bir gözlem kullanırsak ağın daha doğru bir değerlendirmesi elde edilir.

TRAIN İÇİN:

(80)
```
traindata = sigdata(trainidx)
trainlabels = labels(trainidx)
```

readall sonucu çıktı

reonecats sonucu çıktı

VALİDATİON İÇİN:

(81)
```
valdata = sigdata(validx)
vallabels = labels(validx)
```

TEST İÇİN:

(82)
```
testdata = sigdata(testidx)
testlabels = labels(testidx)
```

Deep Learning Ayarı

Sequence Networks'ten → Sequence to Label seçilir.

→ Bir LSTM'in ilk katmanı bir diziyi giriş katmanıdır. Katmanın giriş boyutu, veri setindeki kanal sayısıyla veya bir gözlemdeki satır sayısıyla eşleşmelidir.

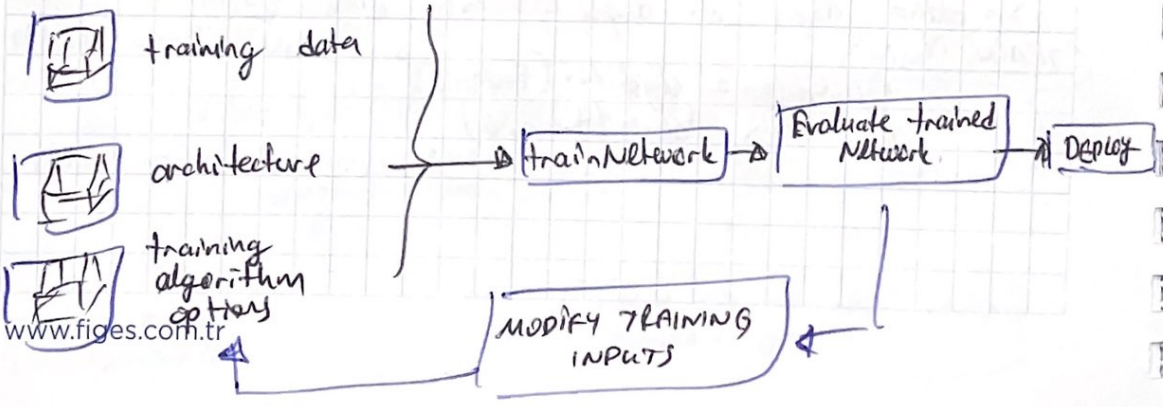
↳ x,y,z → InputSize = 3 olur. (Transpoz almıştık)

→ Fully Connected Layer (Tam Bağlı Katman) girdi olarak sınıf sayısı gerektirir. Kaç farklı desenlik varsa onunla değiştirilir. Örneğin

↳ 0, 0.17, 0.45, 0.73 → OutputSize = 4 olur.

LSTM Network Eğitimi

Workflow (İş akışı)



FİGES

19/12/2023

Derin bir ağı eğitmek için üç bileşen oluşturmanız gerekir
1) Eğitim verisi olarak kullanılmak üzere bilinen etiketlere sahip veri kümesi. Sinyal sınıflandırma problemleri için bu, bir veri deposu veya bir hücre dizisi olarak sağlanabilir
2) Deep Network Designer'da oluşturduğunuz ağ mimarisini temsil eden katmanlar.
3) Eğitim algoritmasının davranışını kontrol eden seçenekler içeren bir değişken.

Ağ eğitildikten sonra test edilmelidir. Sonuçlardan memnun değilseniz, genellikle eğitim girdilerinden bazılarını değiştirir ve ağı yeniden eğitirsiniz.

FONKSİYON:
"trainNetwork" işleri çeşitli girdileri kabul eder. Ancak bu eğitimde aşağıdakiler kullanılacak

(83) `net = trainNetwork (C, Y, layers, options)`

INPUTS:
C: Gözlemlerin hücre dizisi. Her hücre, sütunların zaman adımları ve satırların kanallar olduğu bir gözlem içerir.
Y: C'nin her bir elemanı için karşılık gelen etiket.
layers: Ağ katmanları. Bu katmanlardan birisi 'lstmLayer', diğeri 'bilstmLayer' olmalıdır.
options: trainingOptions fonksiyonu ile oluşturulan eğitim seçenekleri

Ağ eğitimi için aşağıdaki kodu kullanıp algoritmasını belirleyebiliriz.

(84) `opts = trainingOptions ("adam")`

Validasyon seçenekleri eklemek için)

(85) `opts = trainingOptions("adam", "ValidationData", {valdata valLabels}, "Plots", "training-progress")`

"classify" fonksiyonu, yeni sinyallerin sınıflandırma tahminini gerçekleştirmek için eğitilmiş bir LSTM ile kullanılabilir.

```
(86) predictedLabel = classify(net, data)
```

Bir hücre dizisinin her hücresi üzerinde işlem yapmanın hızlı bir yolu bir "cellfun" kullanmaktır.
Veri setindeki gözlemlerin uzunluğunu hesaplamak için, fonksiyon tanıtıcılarını kullanarak uzunluk fonksiyonunu cellfun'a aktarabiliriz.
Bir fonksiyon hakkında tanımlayıcı oluşturmak için fonksiyon ismine "@" koyarız.

```
(87) out = cellfun(@funct, data)
```

↳ Bunu "bar" ile çizdirebiliriz

```
(88) bar(out)
```

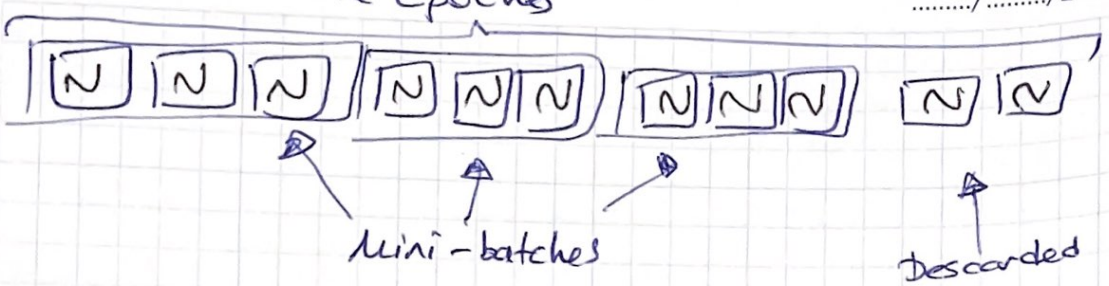
"padsequences" fonksiyonu ile bir sinyal gerektiği gibi doldurulacak veya kesilecektir.

```
(89) sig = padsequences(cells, dim, "Length", 2)
```

Epochs ve Mini-Batches

Her yinelemede, ağırlıkları güncellemek için "mini-batch" olarak bilinen eğitim gözlemlerinin bir alt kümesi kullanılır. Her yinelemede "mini-batches" kullanılır. Eğitim setinin tamamını kullandığında buna "epochs" denir. Derin bir ağ, ağırlıkları birden fazla "epochs" boyunca güncellendiğinde en iyi şekilde eğitilir. Varsayılan "epochs" sayısı 30'dur.

Maksimum "epoch" sayısı (MaxEpochs) ve "mini-batches" (MiniBatchSize), eğitim algoritmanın seçeneklerinde ayarlayabileceğimiz parametrelerdir.



177 eğitim gözleminin olduğu ve mini-batch sayısının 128 olduğu bir senaryo düşünelim. Mini-batch boyutu eğitim gözlemlerinin sayısını eşit şekilde bölmüyorsa trainNetwork kalan gözlemleri atar. Bunu önlemek için mini-batch boyutunu eğitim gözlemlerinin sayısına göre eşit olacak şekilde ayarlayabiliriz.

SIGNAL LABELER

Sinyal veri kümeleri daha önceden etiketlenmemişse sinyalleri etkileşimli olarak etiketlemek için Signal Labeler kullanabilirsiniz. Sinyal özellikleri, bölgeleri ve ilgi çekici noktaları etiketleyebilirsiniz.

Veriler Signal Labeler'da etiketlendikten sonra dışa aktarılır, export edilir. Workspace'de her gözleme ilişkin etiketlerin ilgi alanı tablosu saklanır.

Bu etiketleri derin bir ağ ile eğitmek amacıyla kullanılır ilk adım sinyaller ve etiketler için veri deposu oluşturmaktır.

(90) `[sig, lbl] = createDatastores(lss, "labelname")`

Hücre dizisinden (cell array) bir öğeyi çıkartmak için süslü parantez ({}) kullanabilirsiniz.