T.C.
KTO Karatay Üniversitesi
Mühendislik Fakültesi
Bilgisayar Mühendisliği Bölümü

CE491 - Senior Design Project II
Bitki Tanıma

HAZIRLAYAN
21502774 - Mehmet KALAYCI

DERS DANIŞMANI
Prof. Dr. Abdullah ÇAVUŞOĞLU

KONYA
Ocak, 2018

İçindekiler

1. Giriş

Bitkiler yaşamın sürdürülebilmesi için gereken en önemli kaynaklardandır. Bu kaynakların sağlıklı bir şekilde gelecek nesillere aktarılması, tanımlanmaları, sınıflandırılmaları yaşamımızın sürdürülebilmesinde önemli bir yer teşkil etmektedir. Bazı bitkilerin yaşam döngüsünün devam etmesi, bazılarının ise sahip oldukları özellikler sebebiyle koruma altına alınması ya da yasaklanmasını gerektirmektedir. Ancak bitkilerin tanımlanarak sınıflandırılması işlemi, her bir bitkinin ayrı ayrı incelenip özelliklerinin ortaya çıkarılmasıyla mümkün olacaktır. Bir bitkinin bilgisayar ortamında tanınması işlemi ise o bitkiye ait özellik çıkarımının görüntü işleme teknikleri ile belirlenmesini ve makine öğrenmesi tekniklerini kullanarak eldeki verilerin sınıflandırılmasıyla mümkündür.

Günümüzde veri hacminin çoğalmasıyla birlikte veri işleme tekniklerinde gerçekleşen gelişmelerde de ilerleme kaydedilmiştir. Bu ilerlemelerin sonuçlarından birisi olan derin öğrenmedir. Derin öğrenme yapay sinir ağlarının ileri düzey bir yaklaşımıdır. Derin öğrenmede çok katmanlı bir yapay sinir ağı yapısı mevcuttur. Bu çok katmanlı yapıda görüntüye ait özellik çıkarımları sinir ağı katmanları içerisinde oluşturulmaktadır. Katmanlı yapı öz nitelikleri gizli katmanda çıkararak ağırlık değerliklerini eğitim sürecinde ortaya çıkarmaktadır. Konvolüsyon işlemi ile görüntüye ait kenar ve özellikler belirlenmekte ağdaki diğer katmanlar bu özellikleri alarak bir alt katmana iletmektedir. İlk katman girdi katmanı son katman ise sınıflandırma katmanı olarak bilinmektedir. Bu iki katman arasında yer alan ara katmandaki işlemlerde ise konvolüsyon, maxpooling, dropout, normalizasyon, relu, softmaxlayer, fullconnectedlayer katmanları mevcuttur. Derin öğrenme algoritmaları içerisinde başarım oranının artırılması örnek sayısının çok olması ile ilişkilidir. Örnek sayısı artıkça sonuçlarda daha iyi olduğu görülmektedir. Örnek sayısı arttıkça da veri seti de büyümektedir. Bu sebeple derin öğrenme içerisinde yapılan işlemler yüksek başarıma sahip bilgisayarlarda yapılmaktadır. Yüksek miktarda veri olması sebebiyle ağın eğitim süreci oldukça zaman almaktadır. Sadece işlemci destekli bir bilgisayarda bir ağın eğitim süreci günler hatta haftalar almaktadır. Ancak GPU (Graphichs Processing Unit) kullanılması derin öğrenme ağının eğitim sürecini ciddi oranda kısaltmaktadır [1].

Bu çalışmada bitki tanıma sistemi gerçekleştirmek için konvolüsyonel sinir ağları yöntemi kullanılmıştır. Derin öğrenme işlemlerini gerçekleştirebilmek için günümüzde bir çok kütüphane bulunmaktadır ve bunların çoğu python programlama dili kullanmaktadır. Bunlardan en popülerleri tensorflow, keras, caffe ‘dir. Bu çalışmada basitliği ve anlaşılabilirliği sebebiyle keras tercih edilmiştir. Çalışmanın geri kalan kısmında; ikinci bölümde derin öğrenme, üçüncü bölümde keras kütüphanesi hakkında genel bilgiler ve

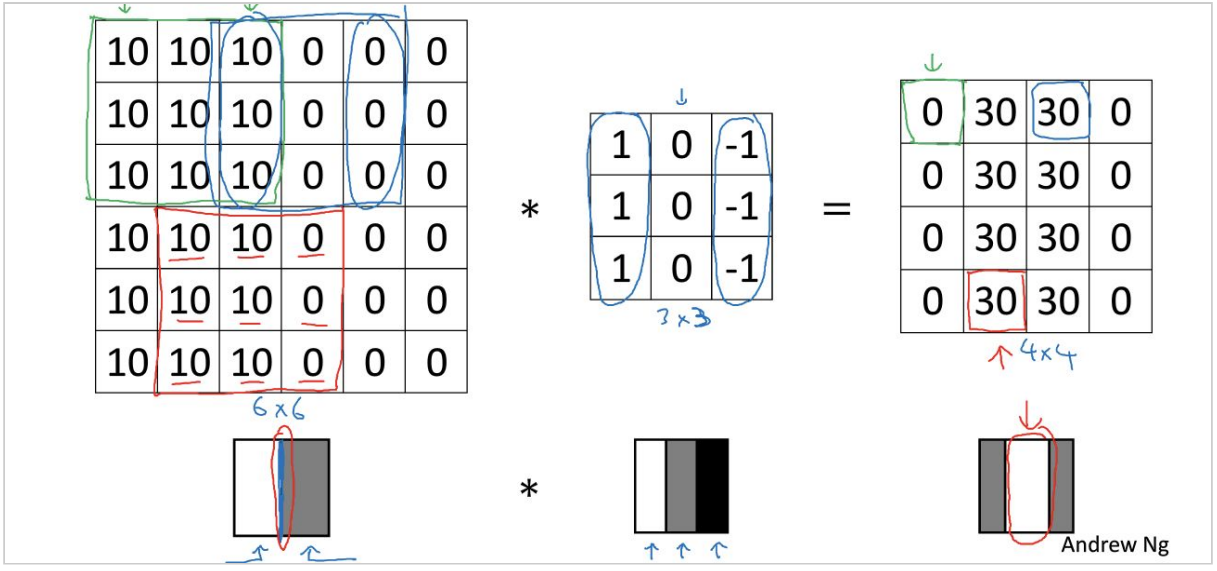
kurulumu, dördüncü bölümde uygulamanın gerçekleştirilmesi ve beşinci bölümde sonuçlara değinilmiştir.

2. Derin Öğrenme

Derin Öğrenme algoritmaları yapay sinir ağlarının yapısal olarak daha karmaşık ve çok katmanlı hali olarak düşünülebilir. Makine öğrenmesi ile bir resimden özellikler çıkarılabilirken, derin öğrenme ile ham verilerden özellik çıkarımı yapılır [2]. Derin Öğrenme algoritmalarının makine öğrenmesindeki algoritmalardan farkı; çok yüksek miktarda veriye ve veriyi işleyebilmek için yüksek hesaplama gücüne sahip donanımlara ihtiyaç duymasıdır.

Çok katmanlı bir yapıya sahip olan derin öğrenme konvolüsyonel sinir ağında her katmanda ayrı bir işlem yürütülerek bir sonraki katmana veriler bırakılır. Her katman kendi işlevini yerine getirir. Derin öğrenme mimarilerinde yer alan katmanlar ve bu katmanların yapmış olduğu işlemler şunlardır.

Konvolüsyon katmanı: Konvolüsyonel sinir ağlarında en temel blok konvolüsyon katmandır. Konvolüsyon, iki kümenin birleştirilmesini sağlayan matematiksel işlemdir. Konvolüsyon filtresi (kernel), girişe uygulanarak özellik haritası (feature map) oluşturulur. Konvolüsyon işlemi görüntü matrisi üzerinde 3x3, 5x5, 7x7, 9x9, 11x11 boyutundaki bir matrisin(kernel) gezdirilmesi ile oluşur. Şekil 2.1 ‘de solda giriş matrisi, ortada konvolüsyon filtresi(kernel), sağda filtre sonucunda oluşan yeni görüntü matrisine ait örnek bulunmaktadır.



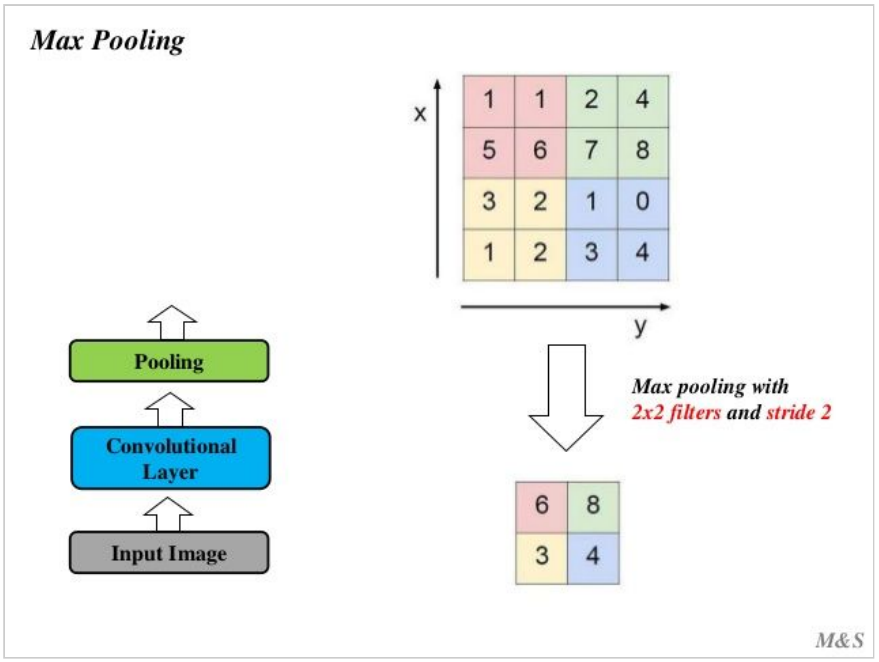
Şekil 2.1: Özellik haritası (feature map) oluşturulması

Girişin üzerinde filtre kaydırılıp konvolüsyon işlemi yapılır. Her eleman bazında matris çarpımı sonucu özellik harita matrisinin karşılıklı elemanını oluşturur.

Gerçek uygulamalarda görüntü 3D gösterilir (yükseklik, genişlik ve derinlik). Derinlik, görüntüdeki renk kanallarını gösterir. RGB için derinlik 3 olarak alınır. Bir giriş üzerinde farklı filtrelere sahip farklı convolution işlemleri yapılabilir. Tüm özellik haritaları birleştirilerek bir özellik haritası elde edilir. Eğer 32x32x3 bir görüntü, 5x5x3 filtre ile konvolüsyon işlemine tabi tutulursa üç tane 5x5x1 matris toplanarak 1x1x1 değer elde edilir. Elde edilen özellik haritası 32x32x1 boyutunda olur. Eğer 10 farklı filtre kullanılırsa, konvolüsyon katmanı 32x32x10 boyutunda olur. Konvolüsyonel sinir ağlarında konvolüsyon operatörünün sonucunu aktivasyon fonksiyonuna gönderir.

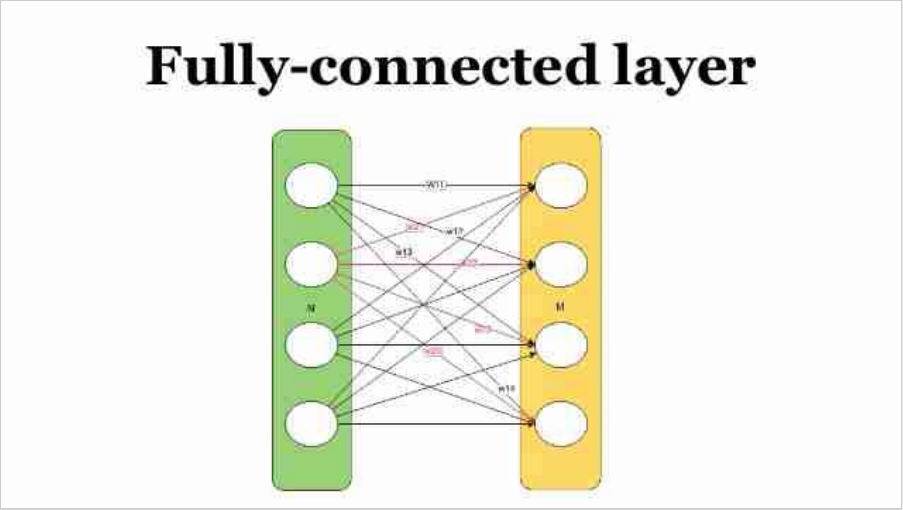
Konvolution filtresinin hareket adım boyutuna **stride** denir. Hareket adım boyutu arttıkça, özellik haritası küçülür. Giriş ile aynı boyutta özellik haritası oluşturmak için kullanılan kavrama ise **padding** denir. Girişin etrafına 0 değerine sahip hücreler padding olarak eklenir.

Havuzlama (Maxpool) Katmanı: Havuzlama katmanı genellikle konvolüsyon işlemi sonrasında uygulanır ve boyut indirgeme yapar. Bu katmanda görüntüde tercihe göre NxN boyutundaki aralıkta bir matris tercih edilir. Bu boş matris görüntü matrisi içerisinde kaydırılarak matris içerisinde yer alan en büyük değeri alır ve yeni bir görüntü matrisi oluşturur. Yeni oluşan görüntü matrisi boyutu küçültülerek yeni görüntü matrisi oluşturulur. Böylelikle ağdaki işlenen veri miktarı da düşürülmüş olur. Şekil 2.2 'de, 2x2 şeklindeki bir matris max pooling yöntemiyle 2x2 'lik maksimum değerin alındığı yeni bir matris oluşturuyor. En yaygın kullanılan max pooling yöntemidir. Yani en konvolüsyon matrisindeki en büyük değeri seçen pooling işlemi.



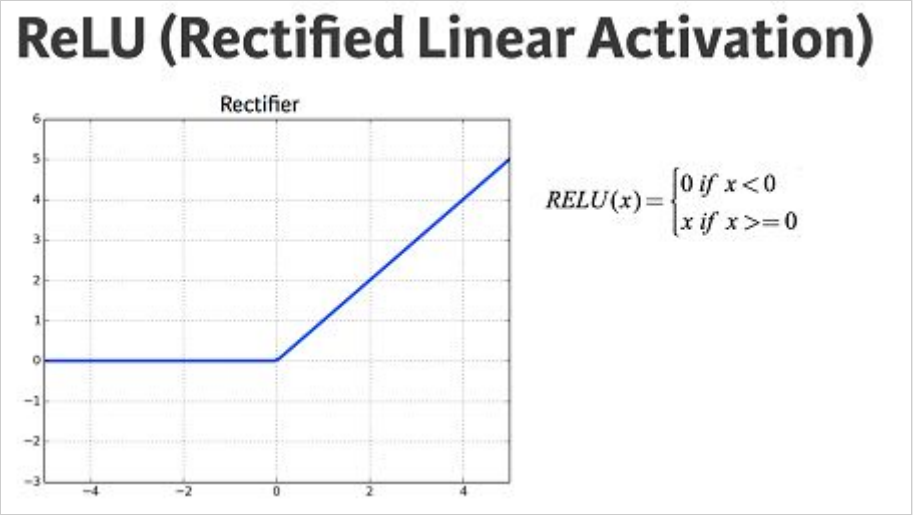
Şekil 2.2: Max-pooling işlemi

Tam Bağlı (Full-Connected) Katmanı: Kendinden önce gelen katmandaki verileri bu katmanda tek boyutlu bir vektör şeklinde yer alır. Nöronlar bu katmanda tam bağlı olarak yer alır. Her nöron kendinden sonra gelen nöronla bağlanır. Bu sebeple tam bağlı katman olarak bilinir. Şekil 2.3 'de tam bağlı katman görüntülenmektedir.



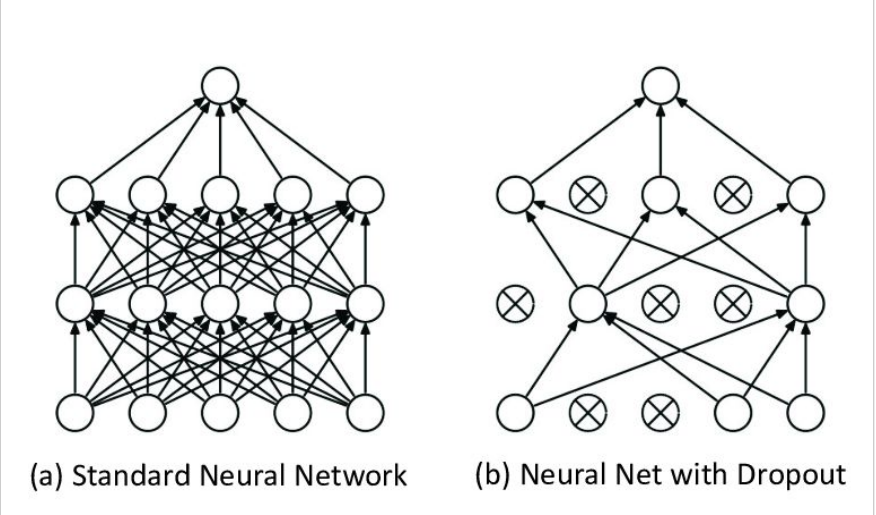
Şekil 2.3: Tam bağlı (full-connected) katmanı

Relu (Rectified Linear Unit) Katmanı: Aktivasyon fonksiyonunun gerçekleştiği katmandır. Bu katmanda elde edilen veriler aktivasyon fonksiyonuna tabi tutulur. Bu katmanda kullanılacak olan pek çok aktivasyon fonksiyonu mevcuttur. Bunlara örnek olarak hiperbolik tanjant, sinüs, step, eşik değer fonksiyonları verilebilir. Derin öğrenme ağlarında genellikle Şekil 2.4 'de gösterilen ReLU aktivasyon fonksiyonu $f(x)$ = max(0, x) tercih edilmektedir.



Şekil 2.4: ReLU aktivasyon fonksiyonu

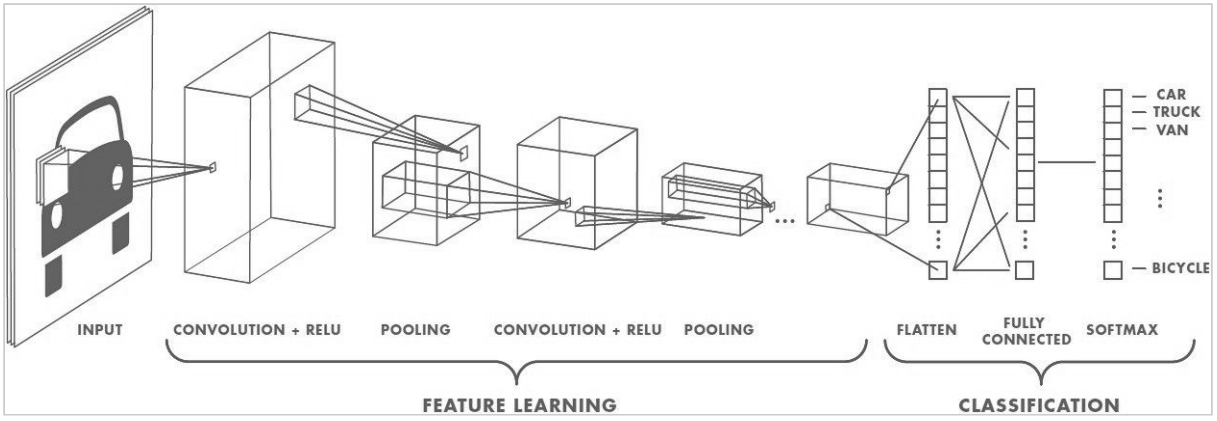
Dropout Katmanı: Derin öğrenme katmanlarında öğrenimi gerçekleştirilmiş bir ağda aşırı öğrenmeyi yani ezberlemeyi ortadan kaldırmak için kullanılan bir katmandır. Sinir ağı içerisinde aşırı öğrenmeyi oluşturan bazı bağlantıları ortadan kaldırma işlemini yaparak ağın ezber yapması engellenir ve performansını artırır. Şekil 2.5 ‘de dropout katmanı örneği görülmektedir.



Şekil 2.5: Dropout katmanı

Normalizasyon (Ölçeklendirme) Katmanı: Derin yapay sinir ağında ağın normalize edilmesi ağın performansını etkilemektedir. Katmanlardan elde edilen verileri düzenli hale getirmektedir. Katmana girdi olan veriler çok büyük ya da çok küçük değerler olabilir. Bu değerlerin normalize edilerek belirli bir aralıkta temsil edilmesi daha sağlıklı olacaktır. Böylelikle hem ağın performansı hem de girdi verilerin ölçeklenerek belirli bir aralıkta temsil edilmesi sağlanır (Ioffe ve Szegedy 2015).

Entropi (Softmax) Katmanı: Önceki katmandan gelen değerleri alarak sınıflandırma işlemi içerisinde olasılıksal değer üretimi gerçekleştirir. Sınıflandırma yaparken hangi sınıfa daha yakın olduğuna dair değer üretir. Derin öğrenme ağı içerisinde katmanda üretilen olasılıksal değeri probalistik hesaplama gerçekleştirerek her bir sınıf için olasılık değeri ortaya koyar (Tang 2013). Bu hesaplama yapılırken cross entropy kullanır. Derin konvolüsyon katmanları arasındaki bağlantılar ve sınıflandırma için yapılacak olan mimari genel olarak Şekil 2.6’da yer almaktadır.



Şekil 2.5: Sınıflandırma için kullanılan mimari

3. Keras Kütüphanesi

Keras, Python programlama dilinde yazılmış ve tensorflow, CNTK veya theano kütüphanelerinin üzerine çıkabilen yüksek düzeyli bir sinir ağları kütüphanesidir. Hızlıca kod geliştirmeye odaklanarak geliştirilmiştir.

- Kolay ve hızlı prototipleme (kullanıcı dostu, modüler ve genişletilebilirlik) sağlar.
- Hem ortak hem de tekrarlayan ağları ve ikisinin kombinasyonlarını destekler.
- CPU ve GPU üzerinde sorunsuz çalışır.
- Keras Python 2.7-3.6. sürümleri ile uyumludur.

3.1. Keras Kütüphanesi Kurulum

Keras kütüphanesi tensorflow, CNTK veya theano kütüphanelerinin üzerine çalışan bir kütüphanedir. Yani keras bunlardan herhangi birine bağımlı vaziyettedir. Ayrıca python programlama dilinde geliştirildiği için keras kütüphanesinin kurulumuna geçmeden önce python'ı kurmalıyız. Python yazılımını https://www.python.org adresinden bilgisayarımıza uygun olan versiyonu seçerek indirebiliriz. Daha sonra python paket yönetim aracı olan pip kullanılarak önce tensorflow sonra keras kütüphanelerini indirmemiz gerekecek. Ancak tüm bu işlemleri yaparken kurulum sırasında bazen sorunlar ve yanlış konfigürasyondan kaynaklı sorunlar meydana gelebilmektedir.

Continum şirketinin çıkardığı yeni python paket sistemi olan anaconda yazılımı bahsedilen problemleri en aza indiren, veri bilimi ve benzeri bilimsel uygulamalar için python kullanmak isteyenlere hazırlanmış tümleşik bir python dağıtımıdır. Anaconda, veri bilimi, yapay zeka gibi konularda sıkça kullanılan kütüphanelerin yanı sıra jupyter notebook ve spyder gibi python editör araçlarını da barındırır. Programı https://www.anaconda.com/download/ adresinden kendi işletim sistemimize uygun olanı

seçerek indirip herhangi bir ayara gerek kalmadan kurabiliriz. Anacondayı kurduğunuzda sisteminizde python, jupiter notebook ve spyderla editörleriyle beraber numpy ve pandas gibi temel veri işleme kütüphaneleri de kurulmuş olacaktır. Bu projede keras kütüphanesi kullanılırken, tensorflow altyapısı tercih edilmiştir. Bu yüzden anaconda prompt aracını kullanarak öncelikle tensorflow kurulacak daha sonra keras kütüphanesi kurulacaktır. TensorFlow'u bir Anaconda ortamında kurmak için aşağıdaki adımları izleyin:

1. Aşağıdaki komutu çağırarak tensorflow adında bir conda ortamı oluşturun:
   C:> conda create -n tensorflow pip python = 3.5
2. Aşağıdaki komutu vererek conda ortamını etkinleştirin:
   C:> activate tensorflow
   (tensorflow) C:> #komut aracı değişmeli
3. TensorFlow'u conda ortamınıza kurmak için uygun komutu verin. TensorFlow'un sadece CPU sürümünü yüklemek için aşağıdaki komutu girin:
   (tensorflow) C:> pip install --ignore-installed --upgrade tensorflow
4. TensorFlow'un GPU sürümünü yüklemek için, aşağıdaki komutu girin (tek bir satırda):
   (tensorflow) C:> pip install --ignore-installed --upgrade tensorflow-gpu

Yukarıdaki adımlar uygulanarak tensorflow kütüphanesi anaconda ortamına yüklenir. Yukarıdaki adımlardan 3-4 adımlarından herhangibiri seçilmelidir. Bu adımda derin öğrenme ağımızı CPU mu GPU mu kullanarak eğiteceğimizi seçmiş oluyoruz. Tensorflow kurulduktan sonra anaconda prompt'a aşağıdaki komut yazılarak keras kütüphanesi paket yöneticisi tarafından otomatik olarak yüklenir.

1. conda install -c conda-forge keras

Yukarıdaki işlemler başarıyla gerçekleştirildiği takdirde, derin öğrenme kodlarımızı yazmaya hazırız. Burada istenilen herhangi bir python kod editör yazılımı kullanılabilir.

3.2. Keras API Yapısı [3]

Keras da iki ana API yapısı vardır. Kuracağınız modelleri bu iki yapıdan birini kullanarak tasarlayabiliyoruz. Sequential yapıda modellerimiz katmanlar şekilde tasarlamak zorundayız. Fakat kullanımı daha sade ve anlaşılır bir yapıdır. Functional yapıda ise fonksiyonlar şeklinde tasarlanıyor. Çok daha esnek ve gelişkin modeller tasarlamamıza imkan sağlıyor. Aşağıda örnek bir keras sequential modeline ait kodlar ve açıklamaları bulunmaktadır.

Sequential API: Katmanlar şeklinde modelimizi oluşturacağımız bir yapı olduğunu söyledik. Genel Kullanımı şekli aşağıdaki gibidir.

```
model = Sequential()
model.add(Layer1(..., input) )
....
model.add(LayerN(...) )
model.add(Dense(output))
model.add(Activation(...))
model.compile(...)
model.fit(...)
model.save('model.h5')
model = load_model('model.h5')
val = np.array([[5,158,84,41,210,39.4,0.395,29]]) # tahmin edilecek nump dizisi verisi
score = model.predict(val)
```

Yukarıda basit şekilde keras sequential model yapısına ait kodlar bulunmaktadır. Bu kodlardaki sequential ifadesi anlamından da çıkarılacağı üzerire modelimizin ardışıl katmanlardan oluştuğunu ifade etmektedir.

Giriş (input): Model dediğimiz yapıda giriş katmanı verimizi vermek için kullanılacağından. Verinin giriş büyüklüğünün belirtilmesi zorunludur. Diğer katmanlar kendilerinden önce gelen katmandan çıkan veriyi aldığından onlar için belirtilmesine gerek yoktur. Giriş Veri büyüklüğü bir kaç değişik şekilde yapılabilir. Aşağıda giriş verisinin tanımlamalarına ait örnek kodlar bulunmaktadır.

1. model.add(Dense(64, input_dim=20,...))
2. model.add(Convolution2D(.... input_shape=(3, 100, 100)))
3. model.add(Dense(32, batch_input_shape=(None, 784)))

#Burada batch (yığın halinde rastgele veri kümesi) boyutu belirtilmediği için, model herhangi bir boyuttaki yığını işleyecek.

4. model.add(Embedding(..., input_length=maxlen))

Çıkış (output): Çıkış katmanında da sonuçları almak için çıkış büyüklüğü belirlenmelidir. Çıkış büyüklüğü genellikle son aktivasyon katmanından önceki katmanın çıkış boyutudur.

1. model.add(Dense(10))

2. model.add(Activation('softmax'))

Mesela bu örnekte 10 adet çıkış değeri olması gerektiği belirtiliyor.

Compile: Model tanımlaması yapıldıktan sonra. Eğitim öncesi bu fonksiyonu çağırmak zorundayız. Bu fonksiyon ile ağın eğitilmesi için kullanılacak optimizasyon fonksiyonu eğitimin durumunu ve ölçümünü veren parametreler verilir.

1. model.compile(**optimizer**=... , **loss**=..., **metrics**=[...])

Optimizasyon Fonksiyonları: Matematiksel fonksiyonların minimum değerlerini bulmaya çalışan optimizasyon algoritmaları, mühendislikte her yerdedir. Diğer şeylerin yanı sıra, tasarım zorunluluklarını değerlendirmek, kontrol sistemlerini değerlendirmek ve verideki kalıpları bulmak için kullanılırlar. Optimizasyon fonksiyonunu optimizer paremetresine iki şekilde eşitleyerek belirtiriz.

1. Optimizasyon fonksiyonun ismini string olarak verip varsayılan parametreleri ile çağırarak.
2. Kendi istediğimiz parametrelerle optimizasyon nesnelerinden birini oluşturarak.

Örnekleyelim:

1. # Optimizer adını yazarak belirtirsek: varsayılan parametreler kullanılacak
   model.compile(loss='mean_squared_error', optimizer='sgd')
2. # Ancak burada optimizer parametrelerini belirtiyoruz
   sgd = SGD(lr=0.01, decay=1e-6, momentum=0.9, nesterov=True)
   model.compile(loss='mean_squared_error', optimizer=sgd)

Kullanabileceğimiz Optimizasyon metotları şunlardır.

1. 'sgd': SGD
2. 'rmsprop': RMSprop
3. 'adagrad': Adagrad
4. 'adadelta': Adadelta
5. 'adam': Adam
6. 'adamax': Adamax
7. 'nadam': Nadam

Loss Fonksiyonları: Ağın eğitimi esnasında ağın ileri doğru çalışıp ürettiği çıkış değerleri ile gerçek çıkış değerlerini karşılaştırıp. Ağın eğitilmesi için geri besleme değeri üreten fonksiyonlardır. Hali hazırda tanımlı fonksiyonlar aşağıdakilerdir.

1. mean_squared_error / mse

2. mean_absolute_error / mae
3. mean_absolute_percentage_error / mape
4. mean_squared_logarithmic_error / msle
5. squared_hinge
6. hinge
7. binary_crossentropy: logloss olarak da bilinir.
8. categorical_crossentropy: multiclass logloss olarak da bilinir.
9. sparse_categorical_crossentropy
10. kullback_leibler_divergence / kld
11. poisson: (predictions - targets * log(predictions)) ortalamasıdır.
12. cosine_proximity

Kullanımı model.compile(loss='categorical_crossentropy',...) şeklindedir. Uygulamanıza özel loss fonksiyonuda yazabilirsiniz

Metric: Compile fonksiyonu için gireceğimiz son parametre metric parametresidir. Fakat Hali hazırda kodlanmış alternatif çeşitler yoktur yegane seçeneğiniz 'accuracy' dir. Size 0-1 arasında bir doğruluk değeri gösterir. Her sınıf için aynı sayıda verimiz varsa oldukça başarılı bir ölçüttür. Fakat her sınıf için çok farklı sayıda örnek varsa çok başarılı bir gösterge olmaz. Model.compile(optimizer=... , loss=..., metrics=['accuracy']) şeklinde kullanılır.

Eğitim Fonksiyonu (Fit): Yapısı tanımlanmış ve compile işlemi yapılmış ağların eğitimi için model sınıfının fit fonksiyonu çağrılır. Önce fonksiyonun tanımlamasına bakalım.

fit(x, y, batch_size=32, nb_epoch=10, verbose=1, callbacks=[], validation_split=0.0, validation_data=None, shuffle=True, class_weight=None, sample_weight=None)

Ağı sabit bir iterasyon sayısınca eğitir.

Parametreler

x: Giriş verisi numpy dizisi ya da numpy dizilerini tutan liste (Eğer ağın çoklu girişi varsa).
y: Giriş verisine ait etiketler numpy dizisi
batch_size: Bir ileri çalışma ve geri yayılma esnasında kullanılacak örnek sayısı tamsayı
nb_epoch: iterasyon sayısı Bütün veri setinin kaç kez tekrar eğitimde kullanılacağı
verbose: eğitim esnasında verilecek ilerleme bilgisini ayarlayan parametre

0 : stdout a bir çıkış yok

1 : ilerleme çubuğuyla göster '[=====>.....]'

2 : her iterasyon için tek ilerleme çubuğu gösterir

callbacks: Eğitim esnasında çalıştırılan fonksiyonlar bakınız callbacks

validation_split: (0. < x < 1). bir oran da verinin bir kısmını eğitimde kullanmayıp doğrulama için kullanılır. Mesela 0.2 demek Eğitim verisinin %20 sinin doğrulama işlemi için kullanılacağı anlamına gelir.

validation_data: Doğrulama için eğitim verisinden ayrı olarak veride verebiliriz bu şekilde tuple tipinde (x, y) Eğitim verisindeki Gibi X giriş Y etiketleri tutan numpy dizileridir.

shuffle: Her iterasyonda eğitim verisinin sırasının karıştırılmasını sağlar true ise

class_weight: Eğitim esnasında loss hesaplanırken her sınıf için ayrı ağırlık parametresi dict tipinde

sample_weight: Eğitim veri setinde loss hesabında her örnek için ayrı bir kat sayı kullanımına imkan verir.

Save: Bu metot ile modelin eğitilmesinden sonra oluşturulan ağırlık değerleri ve model yapısını .h5 uzantılı olarak kaydediliyor. H5py paketi, HDF5 ikili veri formatına bir Pythonic arabirimidir. H5 büyük miktarda sayısal veri depolamanıza ve bu verileri NumPy'den kolayca değiştirmenize izin verir.

Model_load: Model ve ağırlık verileri kaydedilen dosyalardan yüklenir ve yeni bir model oluşturulur.

Predict: Giriş örnekleri için çıktı tahminleri üretir.

Bu bölümde keras kütüphanesine ait sequential model yapısını ve burada kullanılan temel metotları işledik. Burada keras kütüphanesine ait tüm fonksiyonların ve parametrelerin detaylı bir şekilde verilmesi hem konu dahilinde hem pek mümkün değildir. Daha fazla bilgiye keras dokümantasyonun yer aldığı şu adresten https://keras.io erişilebilir.

4. Bitki Tanıma Uygulaması

Bitki tanıma uygulaması derin öğrenme kullanılarak oluşturulan veri kümesinin eğitilmesiyle python programlama dilinde web, mobil ve masaüstü ortamlar için geliştirilmiştir. Web ortamında heroku bulut servisi (https://heroku.com/) kullanılmış ve yazılan web servis aracılığı ile önceden eğitilmiş bir modelin ağırlıkları kullanılarak tanıma işlemi gerçekleştirilmiştir. Mobil ortamda ise bu web servis kullanılmış ve android telefondan çekilen görüntü verisi bu servise gönderilerek alınan yanıt kullanıcıya sunulmuştur. Masaüstü

ortamda ise yazılan python betiği çalıştırıldığında video içerindeki karelerin eğitilmiş ağ üzerinde denenmesi yöntemiyle çalışmaktadır.

4.1. Verilerin Hazırlanması ve Veri Genişletme (Data Augmentation)

Bir ve ikinci bölümlerde derin öğrenme ağlarının çok fazla veriye ihtiyaç duyduğu ve bu yüksek miktardaki verileri işlemek için yüksek işlem kapasitesine sahip grafik işlemcilerinin kullanıldığından bahsetmiştik. Verilerin toplanma işleminde zaman kaybetmemek için derin öğrenme ağınının kullanımı gösteren tensorflow kütüphanesinin sitesindeki bu linkten erişebileceğiniz http://download.tensorflow.org/example_images/flower_photos.tgz hazır verilerden bir kısmı kullanılmıştır. Bahsedildiği gibi derin öğrenme ağının çok miktardaki veriyi işlemesi için yüksek işlem gücüne sahip grafik işlemcisine gerek duyulmaktadır. Ancak bu donanıma tablo 4.1 'deki özelliklere sahip işlemci kullanıldı. Bu nedenle zamandan tasarruf sağlamak için bu veri setinde yer alan papatya (daisy) ve laleler (tulips) verileri kullanıldı. Ancak derin öğrenme ağına ne kadar çok veri sağlanırsa o kadar iyi bir sonuç elde edeceğimizden dolayı keras kütüphanesinin sağladığı keras.preprocessing.image altındaki ImageDataGenerator isimli veri genişletme (data augmentation) modülü ile veri setinde yer alan her bir resmin 32 adet farklı bir görüntüsü oluşturulmuştur. Bu görüntüler keras tarafından tablo 4.2 'deki parametlere göre otomatik olarak rastgele değerlerle oluşturulur.

Özellik Adı	**Değer**
İşlemci Adı	Intel® Core™ i5-3230M CPU @ 2.60GHz
Threading	1 CPU - 2 Core - 4 Threads
Frekans	3975.44 MHz (30 * 99.76 MHz)
Ön Bellekler (Caches)	L1D : 32 KB / L2 : 256 KB / L3 : 3072 KB
Caches Assoc.	L1D : 8-way / L2 : 8-way / L3 : 12-way

Tablo 4.1: İşlemci özellikleri

Parametre Adı	**Değer**	**Açıklama**
rescale	1./255	Yeniden ölçeklendirme faktörü.
shear_range	0.2	Rastgele kesme dönüşümleri uygulamak içindir.

zoom_range	0.2	Resimlerin içinde rastgele yakınlaştırma içindir.
horizontal_flip	True	Görüntülerin yarısını rastgele olarak ters çevirmek içindir.
rotation_range	40	Resimlerin rasgele döndürüleceği bir aralık derece (0-180) cinsinden bir değerdir.
width_shift_range	0.2	Resimleri dikey veya yatay olarak rastgele çeviren aralıklardır (toplam genişliğin veya yüksekliğin bir kısmı olarak).
height_shift_range	0.2	
fill_mode	nearest	Döndürülen veya genişlik/yükseklik kaydırması sonrasında yeni oluşturulan pikselleri doldurmak için kullanılan stratejidir.

Tablo 4.2: ImageDataGenerator modülü için sağlanan parametreler

4.2. Ağın Eğitilmesi ve Modelin Kaydedilmesi

Gerçekleştirilen uygulamada toplamda iki adet sınıf(papatya=0, laleler=1) kullanılmıştır. Genel olarak evrişimsel sinir ağının eğitilmesi aşağıdaki adımlarda belirtilmiştir [4].

1. Öncelikle evrişimsel sinir ağı modeli oluşturulur. Bu modelde konvolüsyon katman sayısı, havuzlama katman sayısı, tam bağlantılı katman sayısı ve sınıflandırma katmanı belirlenir. Bu katmanların sıralanması ve adetleri tasarımcıya özgüdür.
2. Model oluşturulduktan sonra başlangıç değişkenleri tanımlanır. Bu değişkenler temel olarak filtre boyutları, filtre sayısı ve adım kayma miktarı olarak sıralanabilir. Ayrıca her bir filtre için 0-1 aralığında gelişigüzel değerler atanır. Keras ile bu işlem modelin katmanlarına parametre olarak verilir, bazı değerlerin girilmemesi durumunda keras tarafından varsayılan parametre değerleri kullanılır. Örneğin: Stride kernel filtresinin adım sayısı gibi.
3. Oluşturulan model giriş verisi olarak eğitim setinden bir görüntü verilir. Bu görüntü ağdaki katmanlardan geçirilerek bir sonuç değeri elde edilir. Bu aşamaya ileri besleme denir. İleri beslemede her katmanda her bir filtrenin ağırlıkları ile görüntüdeki piksel değerleri çarpılıp bunların toplamı alınarak bir sonraki katmana aktarılır.
4. Ağın üretmiş olduğu sonuçlar ile hedef sonuçların arasındaki hata değeri istenen algoritmaya göre bulunur.
5. Elde edilen hata değerinin ağdaki bütün ağırlıklara dağıtılması gerekmektedir. Bu işlem için geriye yayılım algoritması(GYA) kullanılır. GYA ’da her bir ağırlığın toplam hataya olan etkisinin hesaplanması için istenen bir optimizasyon algoritması kullanılır.

Buradaki ağırlıkların güncellenmesiyle ağın çıkışındaki değeri düşürülmeye çalışılır. GYA'da her bir ağırlığın değeri Gradyan iniş yöntemine göre hesaplanırken kısmi türev kullanılır ve Zincir kuralına göre yapılır.

6. Eğitim kümesindeki bütün görüntüler için Adım 3-5 tekrar edilir.

Projede kullanılan derin öğrenme ağının modeli şekil 4.1 'de verilmiştir. Oluşturulan model giriş olarak 150x150 RGB resimleri 3x3 'lük kernel filtresinden geçirerek işleme tabi tutmaktadır. Bu işlem aşağıdaki şu komut ile;

- model.add(Conv2D(32, (3, 3), input_shape = (150, 150, 3), activation = 'relu'))

gerçekleştirilmektedir. Şekil 4.1 'deki modele karşılık gelen kodlar ise aşağıdaki gibidir.

```
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_9 (Conv2D)            (None, 148, 148, 32)      896
_________________________________________________________________
max_pooling2d_9 (MaxPooling2 (None, 74, 74, 32)        0
_________________________________________________________________
conv2d_10 (Conv2D)           (None, 72, 72, 32)        9248
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 36, 36, 32)        0
_________________________________________________________________
conv2d_11 (Conv2D)           (None, 34, 34, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 17, 17, 64)        0
_________________________________________________________________
flatten_3 (Flatten)          (None, 18496)             0
_________________________________________________________________
dense_5 (Dense)              (None, 128)               2367616
_________________________________________________________________
activation_7 (Activation)    (None, 128)               0
_________________________________________________________________
dropout_3 (Dropout)          (None, 128)               0
_________________________________________________________________
dense_6 (Dense)              (None, 1)                 129
_________________________________________________________________
activation_8 (Activation)    (None, 1)                 0
=================================================================
Total params: 2,396,385
Trainable params: 2,396,385
Non-trainable params: 0
_________________________________________________________________
```

Şekil 4.1: Derin Öğrenme Ağı Modelinin Yapısı

Aşağıdaki verilen kodlarda öncelikle ihtiyaç duyulan python modülleri yüklenmiş ardından model oluşturulmuştur. Bu kodlar şekil 4.1'deki modele karşılık gelmektedir.

```
from keras.models import load_model
from keras.models import Sequential
from keras.layers import Conv2D, MaxPooling2D
from keras.layers import Activation, Dropout, Flatten, Dense
from keras.preprocessing.image import ImageDataGenerator


model = Sequential()
model.add(Conv2D(32, (3, 3), input_shape = (150, 150, 3), activation = 'relu'))
```

```
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Conv2D(32, (3, 3), input_shape = (150, 150, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
model.add(Conv2D(64, (3, 3), input_shape = (150, 150, 3), activation = 'relu'))
model.add(MaxPooling2D(pool_size = (2, 2)))
# the model so far outputs 3D feature maps (height, width, features)
model.add(Flatten())  # this converts our 3D feature maps to 1D feature vectors
model.add(Dense(128))
model.add(Activation('relu'))
model.add(Dropout(0.5))
model.add(Dense(1))
model.add(Activation('sigmoid'))
model.compile(loss='binary_crossentropy', optimizer='rmsprop', metrics=['accuracy'])
```

Bundan sonra verilen kodlarda ise veri setimizde yer alan eğitim ve test verilerinin konumları belirtilerek veri genişletme (data augmentation) işlemi yapılmış, modelimiz eğitilerek model.h5 adında daha sonra kullanılmak üzere kaydedilmiştir.

```
datagen = ImageDataGenerator(rotation_range=40, width_shift_range=0.2, height_shift_range=0.2, rescale=1./255, shear_range=0.2, zoom_range=0.2, horizontal_flip=True, fill_mode='nearest')

train_datagen = ImageDataGenerator(rescale = 1./255, shear_range = 0.2, zoom_range = 0.2, horizontal_flip = True)

test_datagen = ImageDataGenerator(rescale = 1./255)

training_set = train_datagen.flow_from_directory('dataset/train', target_size = (150, 150), batch_size = 32, class_mode = 'binary')

test_set = test_datagen.flow_from_directory('dataset/test', target_size = (150, 150), batch_size = 32, class_mode = 'binary')

model.fit_generator(training_set, steps_per_epoch = 100, epochs = 5, validation_data = test_set, validation_steps = 500)

model.save("model.h5")
```

Ağın eğitimine dair kod çıktısı aşağıdaki verilmiştir. Çıktı incelendiğinde acc (doğruluk) değerimiz giderek artarken ve loss (kayıp) değerimiz ise düşmüştür. Test veri setimizdeki doğruluk oranına dair ilişki ise val_acc ve val_loss değerleriyle gösterilmiştir. Doğruluk oranlarımız genel olarak %80 'in üzerinde seyretmiştir.

```
Found 1232 images belonging to 2 classes.
Found 200 images belonging to 2 classes.
Epoch 1/5        loss: 0.5705 - acc: 0.7141 - val_loss: 0.3414 - val_acc: 0.8500
```

```
Epoch 2/5:      loss: 0.3919 - acc: 0.8337 - val_loss: 0.5226 - val_acc: 0.7098
Epoch 3/5:      loss: 0.3115 - acc: 0.8650 - val_loss: 0.2460 - val_acc: 0.8902
Epoch 4/5:      loss: 0.2837 - acc: 0.8784 - val_loss: 0.2930 - val_acc: 0.8500
Epoch 5/5:      loss: 0.2506 - acc: 0.8928 - val_loss: 0.2610 - val_acc: 0.8752
```

Modelimize herhangi bir papatya ya da lale resmi verdiğimizde tahmin işlemini ise aşağıdaki kodlar ile yapıyoruz. Aşağıdaki tahmin kısmında yer alan kodlarda ise girdi olarak alınan resim1.jpg dosyası öncelikle 150x150 olarak numpy dizisine çevrilip modelimizin predict metoduna veriliyor. Predict metodu ise bu resmin hangi sınıfa ait olduğunu bize bildiriyor.

```
##Prediction Part
import numpy as np
from keras.preprocessing import image

img_pred = image.load_img('resim1.jpg', target_size = (150, 150))
img_pred = image.img_to_array(img_pred)
img_pred = np.expand_dims(img_pred, axis = 0)
result = model.predict(img_pred)

classes = training_set.class_indices

print(classes)
{'daisy': 0, 'tulips': 1}

#list(classes.keys())[list(classes.values()).index(int(0))]

print(result[0][0])
Out: 0.0 # papatya
```

Ancak yukarıda verilen tahmin kısmı test amaçlı olarak kullanılmıştır. Model web servisinde daha sonra kullanılmak üzere kaydedilmiştir.

4.3. Modelin Video İşlemede Kullanılması

Bir önceki konuda 4.2 başlığı altında modelin oluşturulması ve kaydedilmesi aşamaları anlatılmıştı. Ayrıca basit bir şekilde test amaçlı tahmin gerçekleştirilme işleminden de bahsedildi. Bu konu da ise kaydedilmiş bir modelin videolarda kullanılması konusu incelenecektir.

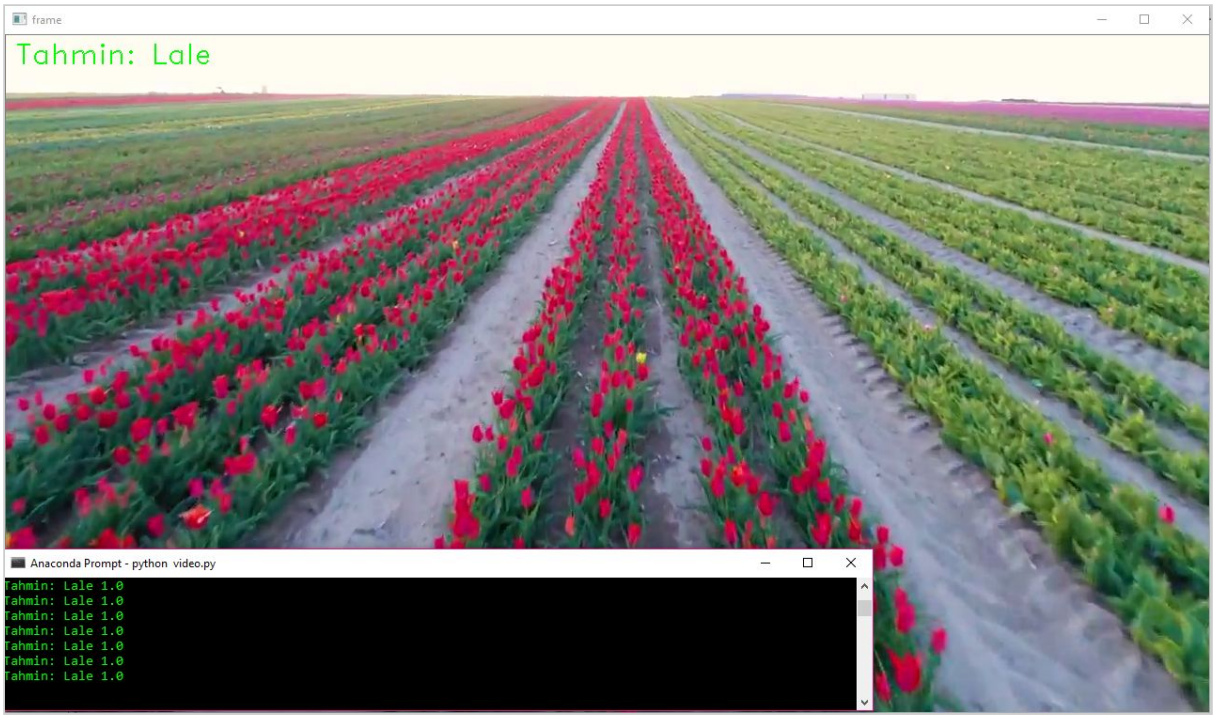
Öncelikle daha önceden kaydedilen model yüklenir ve video kareleri tek tek modele verilerek çıktı hem konsole ekranına hem de video karesine yazılır. Tüm bunları yapan kodlar aşağıdaki gibidir. Kodların çalışmasına ait örnek ekran görüntüsü ise şekil 4.2 ‘deki verilmiştir.

```
import numpy as np
```

```
import cv2
import random
from keras.models import Sequential
from keras.layers import Dense
from keras.models import load_model
from keras.preprocessing import image

cap = cv2.VideoCapture('C:/video.mp4')
frameWidth = int(cap.get(cv2.CAP_PROP_FRAME_WIDTH))
frameHeight = int(cap.get(cv2.CAP_PROP_FRAME_HEIGHT))

model = load_model('model.h5')

while cap.isOpened():
        ret, frame = cap.read()
    cv2.imwrite(filename="alpha.png", img=frame);  # write frame image to file
    # sizing the input to set as input to the convolution network
    img_pred = image.load_img("alpha.png", target_size=(150, 150))
    # convert img to array
    img_pred = image.img_to_array(img_pred)
    img_pred = np.expand_dims(img_pred, axis = 0)
    # prediction part
    result = model.predict(img_pred)
    # write the result to frame's  top left corner
    cv2.putText(img=frame,  text='Tahmin:  {}'.format("Lale"  if  result[0][0]==1 else
    "Papatya"),     org=(10,     30),     fontFace=cv2.FONT_HERSHEY_DUPLEX,
    fontScale=1, color=(0, 255, 0))
    cv2.imshow('frame', frame)
    # write the result to console
    print('Tahmin: {} {}'.format("Lale" if result[0][0]==1 else "Papatya", result[0][0]))
    # quit if press down q key
    if cv2.waitKey(1) & 0xFF == ord('q'):
        break

# release resources
```

cap.release()

cv2.destroyAllWindows()



Şekil 4.2: Video tahmin

4.4. Modelin Web Servisine Dönüştürülmesi

Uygulamanın web üzerinden çalışan bir sistem haline dönüştürülmesi ve ileride android ya da ios gibi farklı işletim sistemi ya da kodlamaya destek sağlaması adına web servisine dönüştürülmesi için https://www.heroku.com adresinden ücretsiz python destekli hosting ve domain hizmeti alınmıştır.

Web servis hizmetini oluşturmak için python da flask isimli modül kullanılmıştır. Uygulamada öncelikle daha önceden oluşturulmuş model yükleniyor ve /predict isimli metoda gönderilen resim dosyası model de tahmin işlemine tabi tutularak sonuçlar json formatında geri döndürülüyor. Uygulamaya ait predict metodunun kodları aşağıdaki verilmiştir.

1. @app.route("/predict", methods=["POST"])
2. def predict():
    2.1. # initialize the data dictionary that will be returned from the
    2.2. # view
    2.3. data = {"success": False}
3.

```
3.1.    # ensure an image was properly uploaded to our endpoint
3.2.    if request.method == "POST" and request.files['image']:
3.2.1.      imagefile = request.files["image"].read()
3.2.2.      image = Image.open(io.BytesIO(imagefile))
3.2.3.      # indicate that the request was a success
3.2.4.      data["success"] = True
3.2.5.
3.2.6.      img_pred = image.img_to_array(image)
3.2.7.      img_pred = np.expand_dims(img_pred, axis = 0)
3.2.8.      result = model.predict(img_pred)
3.2.9.
3.2.10.     data["result"] = result[0][0]
3.2.11.     data["result_str"] = 'Tahmin: {}'.format("Lale" if result[0][0]==1
            else "Papatya"
3.3.    # return the data dictionary as a JSON response
3.4.    return jsonify(data)
```

Heroku da python projesi oluşturulduğunda, projenin ihtiyaç duyduğu modüller requirements.txt adında dosyaya yazılır. Procfile isimli dosyaya web: gunicorn app:app --log-file=- komutu yazılarak uygulama çalıştırılabilir. Oluşturulan web servis hizmetine bu adresten https://bitki-tanima-projesi.herokuapp.com erişilebilir.

5.Sonuçlar

Gerçekleştirilen uygulamada keras kütüphanesi ile 2 sınıfa ait veriler ikili sınıflandırma yöntemiyle evrişimsel sinir ağı kullanılarak model oluşturulmuştur. Oluşturulan model daha sonra kullanılmak üzere kaydedilmiş, video işleme ve web servisi olarak kullanılmıştır. Modelin ikili sınıflandırma yapması girdi verilerinin hangi sınıfa yüzdesel olarak ait olduğunu söyleyememesi bir dezavantajdır. Ayrıca ağın ihtiyaç duyulan görüntülerle eğitilmesi halinde eğitildiği nesneleri tanıması mümkündür. Ağın transfer learning yani eğitilmiş bir ağın son katmanını tekrar eğitmek gibi yöntemlerle başarımını artırmak mümkündür.

Oluşturulan web servis https://bitki-tanima-projesi.herokuapp.com bu adreste yer almaktadır. Projeye ait olan kodlar https://github.com/mehmetkalayci/BitkiTanimaProjesi adresinde yer almaktadır.

KAYNAKLAR

[1] Derin Öğrenme Algoritmalarının Yaprak Sınıflandirma Başarımlarının Karşılaştırılması. (n.d.). Retrieved May 01, 2018, from http://saucis.sakarya.edu.tr/index.php/saucis/article/view/6/6

[2] MATLAB ile Derin Öğrenmeye Giriş (Introducing Deep Learning with MATLAB). (n.d.). Retrieved May 01, 2018, from http://ahmetcevahircinar.com.tr/2017/08/09/matlab-ile-derin-ogrenmeye-giris-introducing-deep-learning-with-matlab/

[3] B. (n.d.). Keras 'a Giriş - 1. Retrieved May 02, 2018, from http://derindelimavi.blogspot.com/2017/01/keras-giris-1.html

[4] Ö, & E. (2017). Derin Öğrenme ve Görüntü Analizinde Kullanılan Derin Öğrenme Modelleri (E., Ed.). GAZİOSMANPAŞA BİLİMSEL ARAŞTIRMA DERGİSİ (GBAD), 6(3), 85-104. Retrieved May 15, 2018, from http://dergipark.gov.tr/download/article-file/380999