Clasificare imagini cu Keras si Tensorflow

January 11, 2021

Catalin Stoean

catalin.stoean@inf.ucv.ro

http://inf.ucv.ro/~cstoean

1 Clasificare de obiecte folosind modele pre-antrenate

Vom utiliza libraria KERAS si modele antrenate anterior pe imagini din ImageNet pentru teste. Ulterior, vom folosi un model preantrenat pentru a invata noi clase in vederea identificarii acestora in imagini noi. ImageNet contine peste 14 milioane de imagini ce fac parte din peste 20 000 de clase (synsets). O buna parte din aceste imagini au servit ca exemple de invatare pentru modele de Deep Learning (DL) ce au "invatat" caracteristici ale fiecarei clase. Printre modelele DL ce pot fi relativ usor folosite prin intermediul librariei Keras (https://keras.io/applications/) sunt: VGG16, VGG19, ResNet50, InceptionV3 etc.

Daca nu au fost instalate anterior, avem nevoie de instalarea keras, tensorflow, h5py. Pentru Tensorflow, la momentul scrierii acestui material, este nevoie ca versiunea de Python sa fie 3.5 sau 3.6 (nu functioneaza cu 3.7). In caz ca aveti alta versiune, aceasta trebuie schimbata.

pip install keras

pip install tensorflow

pip install h5py

Acesta din urma ne trebuie pentru date HDF5.

Pentru a pune o anumita versiune, putem folosi:

pip install keras==2.2.2

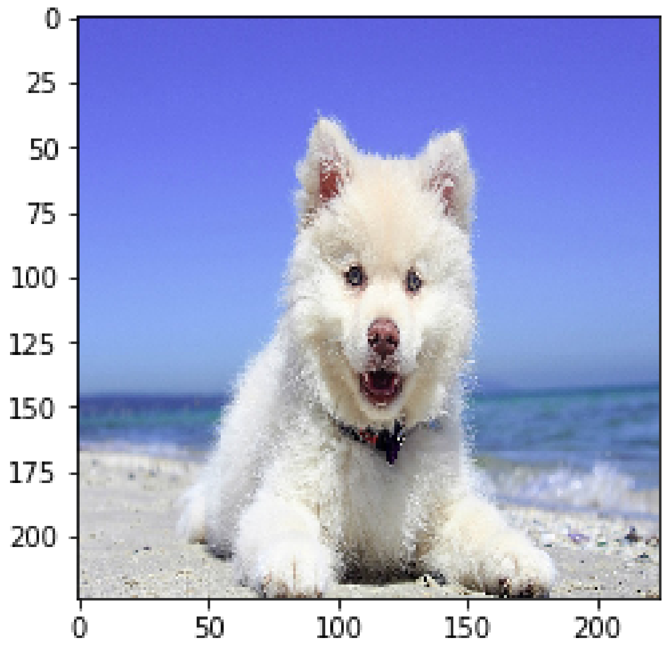
pip install tensorflow==1.8.0

Pentru cei care au placa video nVidia, merita efortul de a instala si CUDA & cuDNN pentru a rula programele folosind GPU in loc de CPU (ruleaza de cca 7 ori mai rapid). Un tutorial video ce poate fi de ajutor gasiti aici:

https://www.youtube.com/watch?v=tPq6NIboLSc

[2]:
```
import keras
import matplotlib.pyplot as plt
%matplotlib inline
from keras.applications.resnet50 import ResNet50
```

```python
from keras.preprocessing import image
from keras.applications.resnet50 import preprocess_input, decode_predictions
import numpy as np


#la prima rulare se descarca modelul
model = ResNet50(weights='imagenet')

calePoza = 'dog-test.jpg'

# Citim poza in format PIL si o redimensionam la 224x224
poza = image.load_img(calePoza, target_size=(224, 224))
plt.imshow(poza)
plt.show()
#transformam imaginea in numpy
x = image.img_to_array(poza)
#modelul accepta numai tensori de marime 4 (batchsize, inaltime, lungime,␣
 ↪canale)
x = np.expand_dims(x, axis=0)
#Preprocesarea extrage media din fiecare canal R, G si B pentru pozele din batch
x = preprocess_input(x)
#Obtinem predictia pentru imagine
preds = model.predict(x)
#Afisam primele 3 optiuni pentru clasa gasita pentru poza
print('Predictia modelului (clasa synset, descriere, probabilitate):\n',
      decode_predictions(preds, top = 3)[0])
```



```
Predictia modelului (clasa synset, descriere, probabilitate):
 [('n02111889', 'Samoyed', 0.77665704), ('n02112018', 'Pomeranian', 0.04919556),
('n02109961', 'Eskimo_dog', 0.027437422)]
```

2 Descarcam imaginile din cateva categorii

imagenetscraper poate fi folosit pentru descarcarea pozelor dupa synset in folderul potrivit. Instalarea se face cu pip install imagenetscraper In Windows mai poate face probleme (a fost si cazul meu), prin urmare am realizat implementarea din celula de mai jos pentru a salva pozele in foldere diferite in functie de synset-urile date.

[ ]:
```python
import cv2
import os
import time
import requests
from requests.adapters import HTTPAdapter
from requests.packages.urllib3.util.retry import Retry

from bs4 import BeautifulSoup
import numpy as np
import requests
import PIL.Image
from PIL import Image
import urllib
import io

#intoarce True daca a putut accesa pagina si False altfel
def exista(path):
    h = requests.head(path, allow_redirects=True)
    header = h.headers
    content_type = header.get('content-type')
    if 'text' in content_type.lower():
        return False
    if 'html' in content_type.lower():
        return False
    return True

# Descarcam imaginea, o convertim in numpy si o transformam in OpenCV
def url_to_image(url):

    image = None
    try:
        resp = urllib.request.urlopen(url)
        image = np.asarray(bytearray(resp.read()), dtype="uint8")
        image = cv2.imdecode(image, cv2.IMREAD_COLOR)
    except urllib.error.HTTPError as exception:
        print(exception)

    return image


idRosii = '07734183'
idRosii2 = '07734292'
idRosii3 = '07734417'
idPiersici = '07751004'
idCirese = '07757511'
idCirese2 = '07757602'


for idCurent in [idPiersici, idCirese, idCirese2, idRosii, idRosii2, idRosii3]:
 ↪#idRosii, idRosii2, idRosii3,
    numeURL = 'http://www.image-net.org/api/text/imagenet.synset.geturls?wnid=n'
    page = requests.get(numeURL + idCurent)#construim pagina cu synset-ul curent
    #page.content = continutul paginii

    '''
    Continutul are link-uri catre poze cu "obiectele" respective.
    De exemplu, la rosii:
    ===
    http://www.reimerseeds.com/images/products/Tomato/
 ↪Burpee_Big_Boy_Tomato_Seeds.jpg
    http://www.istockphoto.com/file_thumbview_approve/2382744/2/
 ↪istockphoto_2382744-isolated-tomatoes.jpg
    http://farm4.static.flickr.com/3255/2787324574_a80540b295.jpg
    http://static.flickr.com/3146/2803582130_c1bc36db42.jpg
    http://www.johnnyseeds.com/images/catalog/product/2373_MED.jpg
    http://www.manikya-veena.com/tomatoes/LIMMONY.jpg
    ...
    ===
    '''

    # BeautifulSoup este o librarie care face citirea din HTML
    soup = BeautifulSoup(page.content, 'html.parser')
    strSoup = str(soup)#obtinem string-uri ce pot fi sparte in bucati

    #obtinem o lista cu url-uri
    split_urls = strSoup.split('\r\n')
    print('Numarul de URL-uri pentru synset-ul {} este {}'
          .format(idCurent, len(split_urls)))
    #Stabilim unde salvam pozele in functie de categoria lor
    numeString = ''
    folder = 'D:/Data2021/'
    if idCurent == idRosii or idCurent == idRosii2 or idCurent == idRosii3:
        numeString = 'rosii'
    elif idCurent == idPiersici:
        numeString = 'piersici'
    elif idCurent == idCirese or idCurent == idCirese2:
        numeString = 'cirese'
    folder += numeString + '/'
    #Daca nu exista folder-ul, il cream
    if not os.path.exists(folder):
        os.makedirs(folder)

    '''
    Luam fiecare URL in parte si salvam poza in folderul curent.
    Vom verifica daca URL-ul este valid, daca pagina contine o poza,
    daca este in format jpg (majoritatea sunt jpg).
    '''

    for i in range(len(split_urls)):
        #Afisam contorul si pagina curenta pentru
        #a o putea verifica in caz de eroare
        print('i = {}, nume = {}'.format(i, split_urls[i]))
        if exista(split_urls[i]):
            print('Exista i = {}, nume = {}'.format(i, split_urls[i]))
            pozaCurenta = url_to_image(split_urls[i])
            if pozaCurenta is not None:#daca s-a citit o poza valida
                cv2.imwrite(folder + numeString
                            + str(idCurent)
                            + 'synset' + str(i)
                            + '.jpg', pozaCurenta)
                print('Poza ' + numeString + str(idCurent)
                      + 'synset' + str(i)
                      + '.jpg este salvata de la ', split_urls[i])
        else:
            print('Sarim peste {} pentru ca {} nu exista'.format(i,␣
↪split_urls[i]))
```

3 Separam imaginile in foldere train/valid

[20]:
```
from shutil import copyfile #pentru copiere de fisiere
from random import sample
'''
Avem acelasi folder in care am salvat pozele mai devreme.
In el vom face un folder train si unul valid.
In train vom copia 67% din poze din fiecare folder in parte.
Le copiem in cate un folder cu acelasi nume.
Restul de 33% vor fi copiate in folderul valid in acelasi mod.
'''
```

```python
folder = 'D:/Data/'
toateFolderele = os.listdir(folder)
#Daca exista si folderele train/valid, acestea sunt eliminate din lista
if 'train' in toateFolderele:
    toateFolderele.remove('train')
if 'valid' in toateFolderele:
    toateFolderele.remove('valid')
print(toateFolderele)
#Luam fiecare folder in parte si copiem pozele in train/valid
for folderCurent in toateFolderele:
    toatePozele = os.listdir(folder + folderCurent)
    print('Folderul {} are {} fisiere'.format(folderCurent, len(toatePozele)))
    nrTrain = int(0.67 * len(toatePozele))
    nrValid = len(toatePozele) - nrTrain
    print('Copiem {} in folderul train si {} in' \
          'folderul valid'.format(nrTrain, nrValid))
    setTrain = sample(toatePozele, nrTrain)
    setValid = list(set(toatePozele) - set(setTrain))
    #cream folderul in train
    #Daca nu exista folder-ul, il cream
    if not os.path.exists(folder + 'train/' + folderCurent):
        os.makedirs(folder + 'train/' + folderCurent)
    #copiem fisierele
    for fileTrain in setTrain:
        copyfile(folder + folderCurent + '/' + fileTrain,
                 folder + 'train/' + folderCurent + '/' + fileTrain)
    print('Am terminat de copiat fisierele pentru antrenament pentru',␣
 ↪folderCurent)
    #Daca nu exista folder-ul, il cream
    if not os.path.exists(folder + 'valid/' + folderCurent):
        os.makedirs(folder + 'valid/' + folderCurent)
    #copiem fisierele pentru validare
    for fileValid in setValid:
        copyfile(folder + folderCurent + '/' + fileValid,
                 folder + 'valid/' + folderCurent + '/' + fileValid)
    print('Am terminat de copiat fisierele pentru validare pentru',␣
 ↪folderCurent)
```

```
['cirese', 'piersici', 'rosii']
Folderul cirese are 614 fisiere
Copiem 411 in folderul train si 203 in folderul valid
Am terminat de copiat fisierele pentru antrenament pentru cirese
Am terminat de copiat fisierele pentru validare pentru cirese
Folderul piersici are 962 fisiere
Copiem 644 in folderul train si 318 in folderul valid
Am terminat de copiat fisierele pentru antrenament pentru piersici
Am terminat de copiat fisierele pentru validare pentru piersici
```

```
Folderul rosii are 1872 fisiere
Copiem 1254 in folderul train si 618 in folderul valid
Am terminat de copiat fisierele pentru antrenament pentru rosii
Am terminat de copiat fisierele pentru validare pentru rosii
```

4 Folosim un model pentru tunificare

[37]:
```
from keras import models
from keras import layers
from keras import optimizers
from keras.applications import VGG16

imDim = 150 #la cat redimensionam toate pozele
#Cu cat marimea pozelor este mai mare, cu atat mai multi parametri are modelul

#la prima rulare se descarca modelul
#redimensionez pozele la 70x70. Unele descarcate anterior au astfel de␣
 ↪dimensiune.
vgg = VGG16(weights='imagenet', include_top=False, input_shape=(imDim, imDim,␣
 ↪3))
'''
Mentinem toate straturile antrenate pe ImageNet, mai putin ultimele 6.
Straturile initiale au invatat caracteristici generale. Pe masura ce
inaintam in straturi, acestea contin caracteristici mai specifice
pentru obiectele ce sunt invatate in problema curenta.
Din acest motiv, eliminam ultimele 6 straturi, urmand a fi
antrenate pentru invatarea unor trasaturi specifice problemei curente.
'''
for layer in vgg.layers[:-6]:#doar ultimele 6 sunt True
    layer.trainable = False

# Afisam toate straturile
for layer in vgg.layers:
    print(layer, layer.trainable)

# Cream un model in care integram si vgg-ul de mai sus
model = models.Sequential()
#Aici se realizeaza agaugarea vgg-ului
model.add(vgg)

# Adaugam alte straturi de final
model.add(layers.Flatten())
model.add(layers.Dense(1024, activation='relu'))
#Dropout reduce o parte din legaturi
model.add(layers.Dropout(0.8))
#La ultimul strat punem numarul de clase (3: rosii, piersici, cirese)
```

```
model.add(layers.Dense(3, activation='softmax'))

# Afisam un sumar al modelului
# Aici vedem si numarul de parametri ce urmeaza sa fie antrenati
model.summary()
```

```
<keras.engine.input_layer.InputLayer object at 0x0000015F0442E108> False
<keras.layers.convolutional.Conv2D object at 0x0000015F0442EB88> False
<keras.layers.convolutional.Conv2D object at 0x0000015F0442E148> False
<keras.layers.pooling.MaxPooling2D object at 0x0000015F0444C808> False
<keras.layers.convolutional.Conv2D object at 0x0000015F0440C508> False
<keras.layers.convolutional.Conv2D object at 0x0000015F04427108> False
<keras.layers.pooling.MaxPooling2D object at 0x0000015F0434ED88> False
<keras.layers.convolutional.Conv2D object at 0x0000015F043F0308> False
<keras.layers.convolutional.Conv2D object at 0x0000015F04345048> False
<keras.layers.convolutional.Conv2D object at 0x0000015F04379208> False
<keras.layers.pooling.MaxPooling2D object at 0x0000015F04335508> False
<keras.layers.convolutional.Conv2D object at 0x0000015F043A2288> False
<keras.layers.convolutional.Conv2D object at 0x0000015F04333F88> False
<keras.layers.convolutional.Conv2D object at 0x0000015F0431BA88> True
<keras.layers.pooling.MaxPooling2D object at 0x0000015F04317908> True
<keras.layers.convolutional.Conv2D object at 0x0000015F04317F08> True
<keras.layers.convolutional.Conv2D object at 0x0000015F04327D08> True
<keras.layers.convolutional.Conv2D object at 0x0000015F0429F8C8> True
<keras.layers.pooling.MaxPooling2D object at 0x0000015F03F1B688> True
WARNING:tensorflow:Large dropout rate: 0.8 (>0.5). In TensorFlow 2.x, dropout()
uses dropout rate instead of keep_prob. Please ensure that this is intended.
Model: "sequential_3"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
vgg16 (Model)                (None, 4, 4, 512)         14714688
_________________________________________________________________
flatten_3 (Flatten)          (None, 8192)              0
_________________________________________________________________
dense_5 (Dense)              (None, 1024)              8389632
_________________________________________________________________
dropout_3 (Dropout)          (None, 1024)              0
_________________________________________________________________
dense_6 (Dense)              (None, 3)                 3075
=================================================================
Total params: 23,107,395
Trainable params: 17,831,939
Non-trainable params: 5,275,456
_________________________________________________________________
```

5 Incarcam datele pentru antrenament si validare

[38]:
```python
from keras.preprocessing.image import ImageDataGenerator
train_datagen = ImageDataGenerator(
      rescale=1./255, #normalizam pozele
      rotation_range=30, #permitem rotatia in poze
      width_shift_range=0.3, #mutarea pe orizontala
      height_shift_range=0.3, #mutarea pe verticala
    fill_mode='nearest')
#normalizam si datele din validare, insa pe acestea nu le modificam altfel
validation_datagen = ImageDataGenerator(rescale=1./255)
'''
 Stabilim cate poze (batchsize) sa primim la fiecare etapa.
 In cazul in care avem eroare de memorie,
 stabilim valori mai mici pentru batchsize.
'''
train_batchsize = 20
val_batchsize = 20

folder = 'D:/Data/'

train_generator = train_datagen.flow_from_directory(
        folder + '/train',
        target_size=(imDim, imDim),
        batch_size=train_batchsize,
        class_mode='categorical')

validation_generator = validation_datagen.flow_from_directory(
        folder + '/valid',
        target_size=(imDim, imDim),
        batch_size=val_batchsize,
        class_mode='categorical',
        shuffle=False)
```

```
Found 2222 images belonging to 3 classes.
Found 1095 images belonging to 3 classes.
```

6 Antrenam modelul

[ ]:
```python
# Compilam modelul
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
# Antrenam modelul
history = model.fit_generator(
      train_generator,
      steps_per_epoch=train_generator.samples/train_generator.batch_size ,
      epochs=20,
      validation_data=validation_generator,
      validation_steps=validation_generator.samples/validation_generator.
 ↪batch_size,
      verbose=1)

# Modelul se poate salva si apoi incarca fara antrenare
model.save('Model_ultimele4.hdf5')
```

7 Daca vrem sa salvam doar cand un model

8 are acuratete pe validare mai buna

[39]:
```
from keras.callbacks import ModelCheckpoint

checkpoint = ModelCheckpoint('bestValModel.hdf5',
                             monitor='val_acc', verbose=1,
                             save_best_only=True, mode='max')

# Compilam modelul, ca mai sus
model.compile(loss='categorical_crossentropy',
              optimizer=optimizers.RMSprop(lr=1e-4),
              metrics=['acc'])
# Antrenam modelul
history = model.fit_generator(
    train_generator,
    steps_per_epoch=train_generator.samples/train_generator.batch_size ,
    epochs=20,
    callbacks=[checkpoint],
    validation_data=validation_generator,
    validation_steps=validation_generator.samples/validation_generator.
 ↪batch_size,
    verbose=1)
```

```
Epoch 1/20
112/111 [==============================] - 44s 389ms/step - loss: 1.0846 - acc:
0.4833 - val_loss: 0.5982 - val_acc: 0.5242

Epoch 00001: val_acc improved from -inf to 0.52420, saving model to
bestValModel.hdf5
Epoch 2/20
112/111 [==============================] - 36s 320ms/step - loss: 0.9196 - acc:
0.5756 - val_loss: 0.4860 - val_acc: 0.7114

Epoch 00002: val_acc improved from 0.52420 to 0.71142, saving model to
bestValModel.hdf5
```

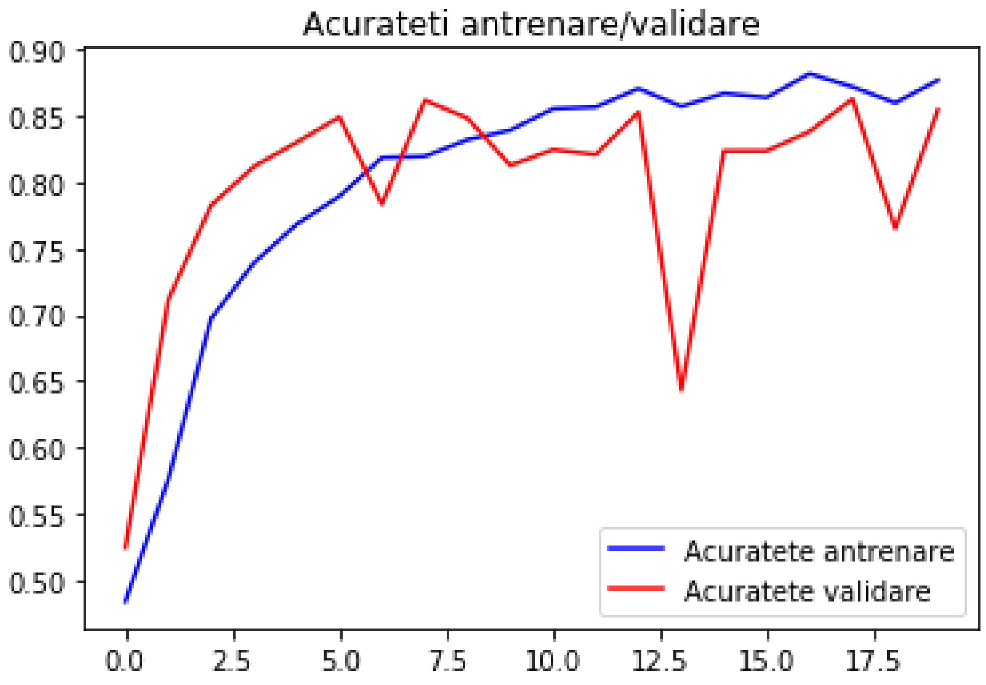
```
Epoch 3/20
112/111 [==============================] - 39s 346ms/step - loss: 0.7481 - acc:
0.6971 - val_loss: 0.2806 - val_acc: 0.7826

Epoch 00003: val_acc improved from 0.71142 to 0.78265, saving model to
bestValModel.hdf5
Epoch 4/20
112/111 [==============================] - 33s 296ms/step - loss: 0.6788 - acc:
0.7390 - val_loss: 0.1729 - val_acc: 0.8119

Epoch 00004: val_acc improved from 0.78265 to 0.81187, saving model to
bestValModel.hdf5
Epoch 5/20
112/111 [==============================] - 34s 308ms/step - loss: 0.5863 - acc:
0.7682 - val_loss: 0.1909 - val_acc: 0.8301

Epoch 00005: val_acc improved from 0.81187 to 0.83014, saving model to
bestValModel.hdf5
Epoch 6/20
112/111 [==============================] - 33s 291ms/step - loss: 0.5758 - acc:
0.7894 - val_loss: 0.2263 - val_acc: 0.8493

Epoch 00006: val_acc improved from 0.83014 to 0.84932, saving model to
bestValModel.hdf5
Epoch 7/20
112/111 [==============================] - 684s 6s/step - loss: 0.4962 - acc:
0.8186 - val_loss: 0.0684 - val_acc: 0.7836

Epoch 00007: val_acc did not improve from 0.84932
Epoch 8/20
112/111 [==============================] - 37s 327ms/step - loss: 0.4934 - acc:
0.8195 - val_loss: 0.2340 - val_acc: 0.8621

Epoch 00008: val_acc improved from 0.84932 to 0.86210, saving model to
bestValModel.hdf5
Epoch 9/20
112/111 [==============================] - 44s 397ms/step - loss: 0.4464 - acc:
0.8321 - val_loss: 0.4064 - val_acc: 0.8484

Epoch 00009: val_acc did not improve from 0.86210
Epoch 10/20
112/111 [==============================] - 35s 309ms/step - loss: 0.4369 - acc:
0.8393 - val_loss: 0.0599 - val_acc: 0.8128

Epoch 00010: val_acc did not improve from 0.86210
Epoch 11/20
112/111 [==============================] - 1556s 14s/step - loss: 0.4142 - acc:
0.8555 - val_loss: 1.1343 - val_acc: 0.8247
```

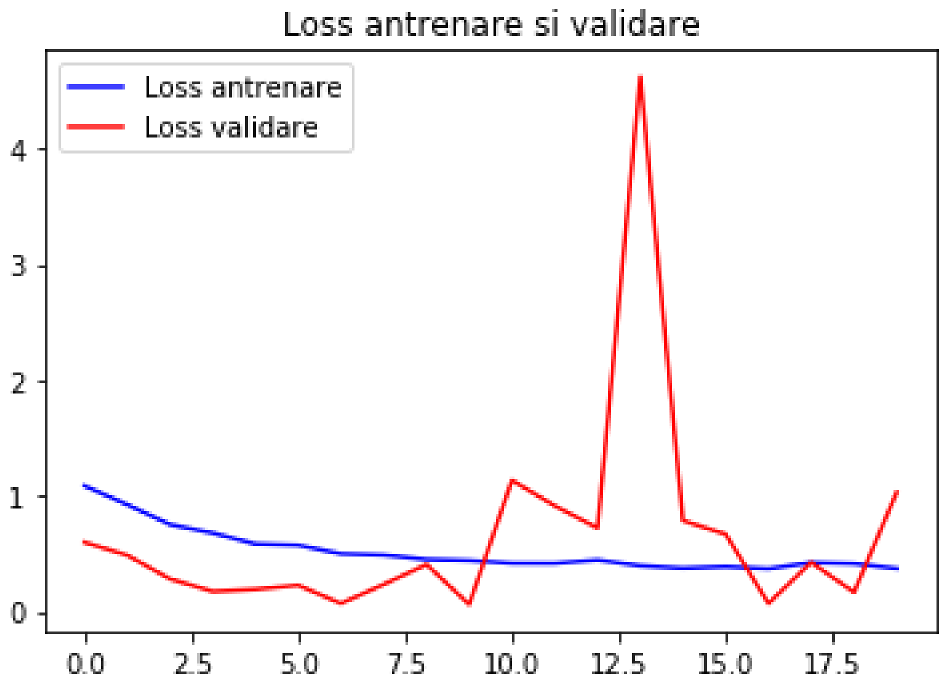
```
Epoch 00011: val_acc did not improve from 0.86210
Epoch 12/20
112/111 [==============================] - 30s 270ms/step - loss: 0.4174 - acc:
0.8569 - val_loss: 0.9129 - val_acc: 0.8210

Epoch 00012: val_acc did not improve from 0.86210
Epoch 13/20
112/111 [==============================] - 32s 289ms/step - loss: 0.4409 - acc:
0.8708 - val_loss: 0.7219 - val_acc: 0.8530

Epoch 00013: val_acc did not improve from 0.86210
Epoch 14/20
112/111 [==============================] - 32s 290ms/step - loss: 0.3972 - acc:
0.8573 - val_loss: 4.6256 - val_acc: 0.6429

Epoch 00014: val_acc did not improve from 0.86210
Epoch 15/20
112/111 [==============================] - 35s 315ms/step - loss: 0.3728 - acc:
0.8672 - val_loss: 0.7843 - val_acc: 0.8237

Epoch 00015: val_acc did not improve from 0.86210
Epoch 16/20
112/111 [==============================] - 30s 270ms/step - loss: 0.4186 - acc:
0.8641 - val_loss: 0.6685 - val_acc: 0.8237

Epoch 00016: val_acc did not improve from 0.86210
Epoch 17/20
112/111 [==============================] - 31s 277ms/step - loss: 0.3720 - acc:
0.8821 - val_loss: 0.0682 - val_acc: 0.8384

Epoch 00017: val_acc did not improve from 0.86210
Epoch 18/20
112/111 [==============================] - 34s 304ms/step - loss: 0.4213 - acc:
0.8722 - val_loss: 0.4251 - val_acc: 0.8630

Epoch 00018: val_acc improved from 0.86210 to 0.86301, saving model to
bestValModel.hdf5
Epoch 19/20
112/111 [==============================] - 32s 289ms/step - loss: 0.4104 - acc:
0.8600 - val_loss: 0.1654 - val_acc: 0.7653

Epoch 00019: val_acc did not improve from 0.86301
Epoch 20/20
112/111 [==============================] - 37s 331ms/step - loss: 0.3686 - acc:
0.8771 - val_loss: 1.0301 - val_acc: 0.8548

Epoch 00020: val_acc did not improve from 0.86301
```

9 Facem un grafic cu acuratetea si loss-ul

10 pe antrenament si validare din timpul antrenarii

[40]:
```
import matplotlib.pyplot as plt
#acuratetea pe antrenare din timpul invatarii
acc = history.history['acc']
#acuratetea pe validare
val_acc = history.history['val_acc']
#loss (eroarea) pe antrenare si validare
loss = history.history['loss']
val_loss = history.history['val_loss']

epochs = range(len(acc))

plt.plot(epochs, acc, 'b', label='Acuratete antrenare')
plt.plot(epochs, val_acc, 'r', label='Acuratete validare')
plt.title('Acurateti antrenare/validare')
plt.legend()

plt.figure()

plt.plot(epochs, loss, 'b', label='Loss antrenare')
plt.plot(epochs, val_loss, 'r', label='Loss validare')
plt.title('Loss antrenare si validare')
plt.legend()

plt.show()
```


Acurateti antrenare/validare
0.90
0.85
0.80
0.75
0.70
0.65
0.60
0.55
0.50
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5
Acuratete antrenare
Acuratete validare


Loss antrenare si validare
Loss antrenare
Loss validare
4
3
2
1
0
0.0
2.5
5.0
7.5
10.0
12.5
15.0
17.5

11 Analizam cateva exemple de poze clasificate gresit

[41]:
```python
import numpy as np
from keras.preprocessing import image
from random import sample

# Luam numele fisierelor din generator
fisiere = validation_generator.filenames

# Luam clasele reale din generator
# calsificariCorecte contine lista tuturor valorilor de clase corecte
calsificariCorecte = validation_generator.classes
print('calsificariCorecte = ', calsificariCorecte)

# Etichetele si valorile asociate
#{'cirese': 0, 'piersici': 1, 'rosii': 2} - in cazul nostru
eticheteValori = validation_generator.class_indices

print('eticheteValori = ', eticheteValori)

#Scoatem potrivirea dintre clase si etichete
valEtichete = dict((v,k) for k,v in eticheteValori.items())
print('valEtichete = ', valEtichete)

#Scoatem predictiile modelului folosind generatorul
predictii = model.predict_generator(validation_generator,
    steps=validation_generator.samples/validation_generator.batch_size,
    verbose=1)
# Adunam in clasePrezise lista tuturor valorilor de clase prezise
clasePrezise = np.argmax(predictii, axis=1)
print('clasePrezise = ', clasePrezise)

# In erori pastram numai acele pozitii ce nu corespund dintre clasificarile␣
 ↪corecte si gresite
erori = np.where(clasePrezise != calsificariCorecte)[0]
print("Numarul de erori este de {} dintr-un total" \
    " de {} imagini".format(len(erori), validation_generator.samples))
```

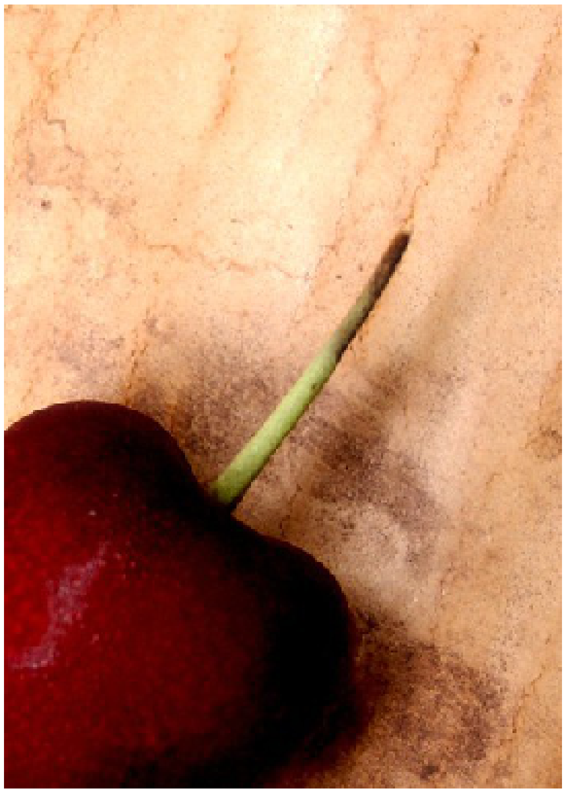
```
calsificariCorecte =  [0 0 0 … 2 2 2]
eticheteValori =  {'cirese': 0, 'piersici': 1, 'rosii': 2}
valEtichete =  {0: 'cirese', 1: 'piersici', 2: 'rosii'}
55/54 [==============================] - 12s 223ms/step
clasePrezise =  [0 0 0 … 1 2 2]
Numarul de erori este de 159 dintr-un total de 1095 imagini
```

11.1 Poze gresite cu probabilitatile pentru clasa gasita

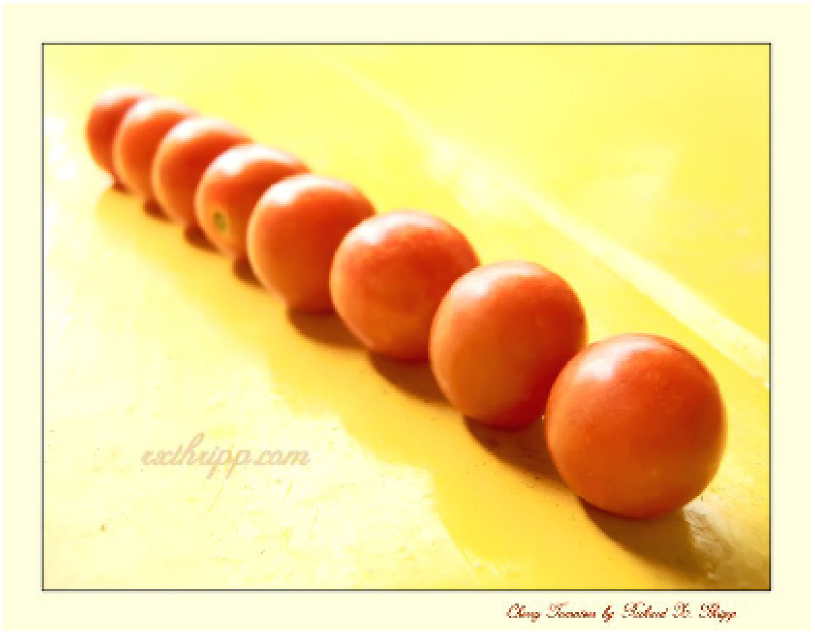
[42]:
```python
# Afisam cateva (3) imagini clasificate gresit

folder = 'D:/Data/'
for i in sample(range(len(erori)), 3):
    #se ia clasa cu probabilitatea cea mai mare
    clasaPrezisa = np.argmax(predictii[erori[i]])
    etichetaPrezisa = valEtichete[clasaPrezisa]

    title = 'Fisierul: {}, Predictia: {}, Probabilitatea: {:.3f}'.format(
        fisiere[erori[i]],
        etichetaPrezisa,
        predictii[erori[i]][clasaPrezisa])

    original = image.load_img('{}/{}'.format(folder + '/valid',␣
↪fisiere[erori[i]]))
    plt.figure(figsize=[7,7])
    plt.axis('off')
    plt.title(title)
    plt.imshow(original)
    plt.show()
```

```
Fisierul: rosii\Tomatoes07734292synset1212.jpg, Predictia: cirese, Probabilitatea: 0.521
```

Fisierul: cirese\Cherries07757602synset286.jpg, Predictia: piersici, Probabilitatea: 0.506



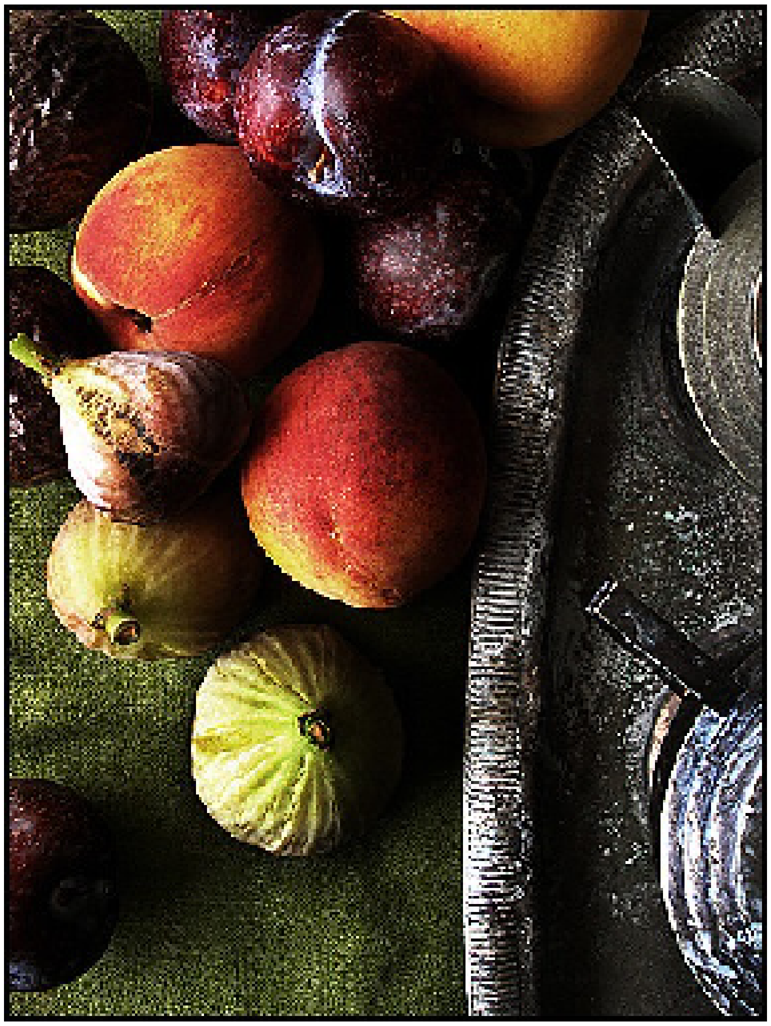
Fisierul: rosii\Tomatoes07734292synset912.jpg, Predictia: piersici, Probabilitatea: 0.970



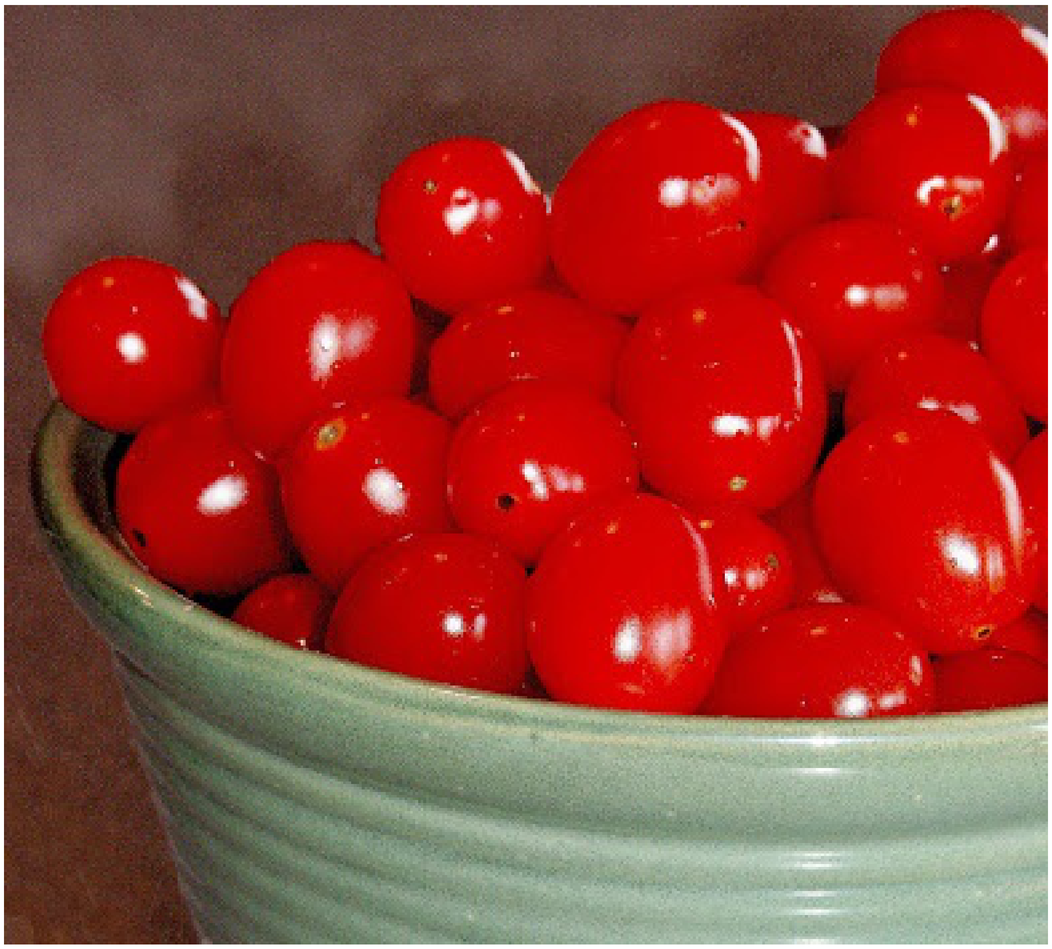
11.2 Poze gresite cu probabilitatile pentru toate clasele

[52]:
```python
# Afisam cateva (3) imagini clasificate gresit
for i in sample(range(len(erori)), 3):
    #se ia clasa cu probabilitatea cea mai mare
    clasaPrezisa = np.argmax(predictii[erori[i]])
    etichetaPrezisa = valEtichete[clasaPrezisa]

    title = 'Fisierul: {}, Predictia: {}\nCireasa: {:.3f}; Piersica: {:.3f};␣
↪Rosie: {:.3f}'.format(
        fisiere[erori[i]],
        etichetaPrezisa,
        predictii[erori[i]][0], predictii[erori[i]][1], predictii[erori[i]][2])

    original = image.load_img('{}/{}'.format(folder + '/valid',␣
↪fisiere[erori[i]]))
    plt.figure(figsize=[7,7])
    plt.axis('off')
    plt.title(title)
    plt.imshow(original)
    plt.show()
```

```
Fisierul: piersici\Peaches07751004synset529.jpg, Predictia: cirese
Cireasa: 0.999; Piersica: 0.000; Rosie: 0.001
```



Fisierul: rosii\Tomatoes07734417synset612.jpg, Predictia: cirese
Cireasa: 0.648; Piersica: 0.000; Rosie: 0.352

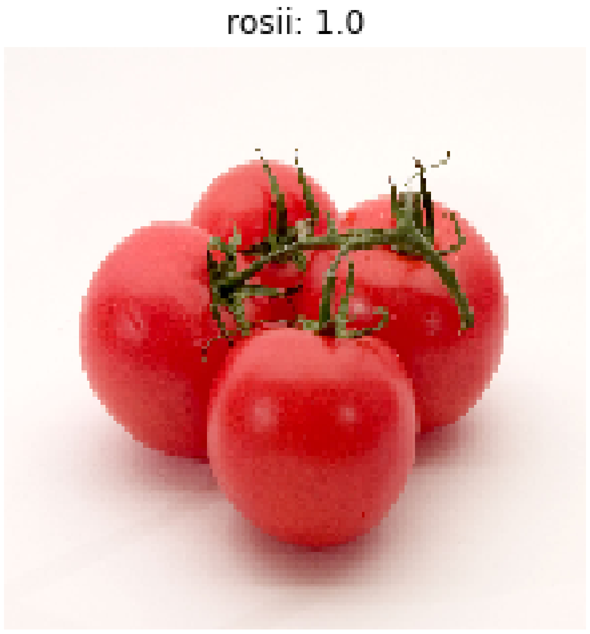


Fisierul: cirese\Cherries07757511synset599.jpg, Predictia: rosii
Cireasa: 0.000; Piersica: 0.000; Rosie: 1.000

12 Aplicam modelul pentru poze noi din fiecare clasa

[48]:
```
#am descarcat primele poze gasite in pe internet cand am cautat cele 3 cuvinte␣
 ↪cheie
pozeNoi = ['piersica.jpg', 'rosie.jpg', 'cireasa.jpg']


for p in pozeNoi:
    img = image.load_img(p, target_size=(imDim, imDim))
    x = image.img_to_array(img)
    x = np.expand_dims(x, axis=0)

    predictie = model.predict(x)

    clasaPrezisa = np.argmax(predictie[0])

    etichetaPrezisa = valEtichete[clasaPrezisa]

    plt.axis('off')
    plt.title(etichetaPrezisa + ': ' + str(predictie[0][clasaPrezisa]))
    plt.imshow(img)
    plt.show()
```

piersici: 1.0


rosii: 1.0

13 Exercitii

1. Modificati parametrii pentru modelul de mai sus pentru a duce acuratetea de clasificarea la minimum 90%.
2. Realizati un model care sa distinga intre 4 tipuri de obiecte.