# 16. Image classification using deep learning

In the previous notebooks, we have mostly focused on the segmentation task, i.e isolating structures in images. Another major image processing task is instead to classify entire images. For example when screening for skin caner, one is not necessarily in segmenting a tumor but rather saying whether a tumor is absent or present in an image.

Deep learning methods have been shown in the past years to be very efficient in this exercise, and many different networks have been designed. A lot of models can be found online, for example on Github. In addition, Keras, a very popular high-level package for machine learning, offers ready-to-use implementations of many popular networks. Those networks have already been trained on specific datasets, but of course one can re-train them to solve other classification tasks. Here we are going to see how to use these Keras implementations.

## 16.1 Importing the model

It is straightforward to import the needed model. Documentations can be found here (https://keras.io/applications/). Here we are using the VGG16 model (https://arxiv.org/abs/1409.1556) that has been trained on the ImageNet dataset, which classifies objects in 1000 categories.

In [1]:
```
from keras.applications.vgg16 import VGG16
from keras.applications.vgg16 import preprocess_input
from keras.applications.vgg16 import decode_predictions

#from keras.applications.xception import Xception
#from keras.applications.xception import preprocess_input
#from keras.applications.xception import decode_predictions

import numpy as np
import skimage
import skimage.io
import skimage.transform
import matplotlib.pyplot as plt
```

```
Using TensorFlow backend.
```

Now we load the model, specifying the weights to be used. Those weights define all the filters that are used in the convolution steps as well as the actual weights that combine information from the output of different filters.

In [2]:
```
model = VGG16(weights='imagenet', include_top=True)
#model = Xception(weights='imagenet', include_top=True)
```

```
WARNING: Logging before flag parsing goes to stderr.
W0123 11:15:52.456051 140581987952384 deprecation_wrapper.py:119] From /u
sr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:
4070: The name tf.nn.max_pool is deprecated. Please use tf.nn.max_pool2d
instead.


Downloading data from https://github.com/fchollet/deep-learning-models/re
leases/download/v0.1/vgg16_weights_tf_dim_ordering_tf_kernels.h5
553467904/553467096 [==============================] - 91s 0us/step
```

We can have a look at the structure of the network:

In [3]:
```
model.summary()
```

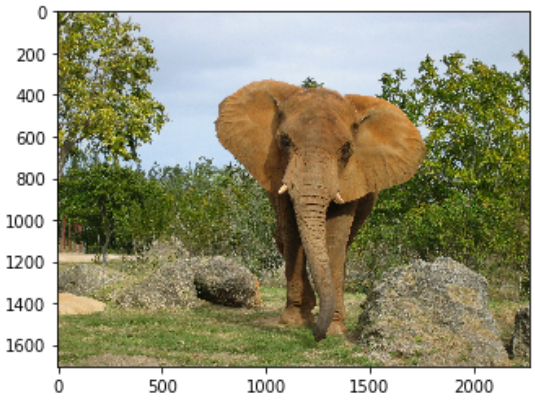
```
Model: "vgg16"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
input_1 (InputLayer)         (None, 224, 224, 3)       0
_________________________________________________________________
block1_conv1 (Conv2D)        (None, 224, 224, 64)      1792
_________________________________________________________________
block1_conv2 (Conv2D)        (None, 224, 224, 64)      36928
_________________________________________________________________
block1_pool (MaxPooling2D)   (None, 112, 112, 64)      0
_________________________________________________________________
block2_conv1 (Conv2D)        (None, 112, 112, 128)     73856
_________________________________________________________________
block2_conv2 (Conv2D)        (None, 112, 112, 128)     147584
_________________________________________________________________
block2_pool (MaxPooling2D)   (None, 56, 56, 128)       0
_________________________________________________________________
block3_conv1 (Conv2D)        (None, 56, 56, 256)       295168
_________________________________________________________________
block3_conv2 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_conv3 (Conv2D)        (None, 56, 56, 256)       590080
_________________________________________________________________
block3_pool (MaxPooling2D)   (None, 28, 28, 256)       0
_________________________________________________________________
block4_conv1 (Conv2D)        (None, 28, 28, 512)       1180160
_________________________________________________________________
block4_conv2 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_conv3 (Conv2D)        (None, 28, 28, 512)       2359808
_________________________________________________________________
block4_pool (MaxPooling2D)   (None, 14, 14, 512)       0
_________________________________________________________________
block5_conv1 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv2 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_conv3 (Conv2D)        (None, 14, 14, 512)       2359808
_________________________________________________________________
block5_pool (MaxPooling2D)   (None, 7, 7, 512)         0
_________________________________________________________________
flatten (Flatten)            (None, 25088)             0
_________________________________________________________________
fc1 (Dense)                  (None, 4096)              102764544
_________________________________________________________________
fc2 (Dense)                  (None, 4096)              16781312
_________________________________________________________________
predictions (Dense)          (None, 1000)              4097000
=================================================================
Total params: 138,357,544
Trainable params: 138,357,544
Non-trainable params: 0
_________________________________________________________________
```

## 16.2 Choosing and adjusting an image

Let's test the network on a simple image of an elephant:

In [4]:
```
image = skimage.io.imread('https://upload.wikimedia.org/wikipedia/commons/1/19/Afrikanische_Elefant%2C_Miami2.jpg')
```

In [5]:
```
plt.imshow(image)
plt.show()
```




Models are always expecting images of a certain size, and with intensities around a given values. This is taken care of here:

In [6]:
```
#adjust image size and dimensions
image_resize = skimage.transform.resize(image,(224,224),preserve_range=True)
x = np.expand_dims(image_resize, axis=0)

#adjust image intensities
x = preprocess_input(x)
```

```
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:105: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:110: UserWarning: Anti-aliasing will be enabled by default in skimage 0.15 to avoid aliasing artifacts when down-sampling images.
  warn("Anti-aliasing will be enabled by default in skimage 0.15 to "
```

## 16.3 Prediction

Finally, we can pass that modified image to the network to give a prediction:

In [7]:
```
features = model.predict(x)
```

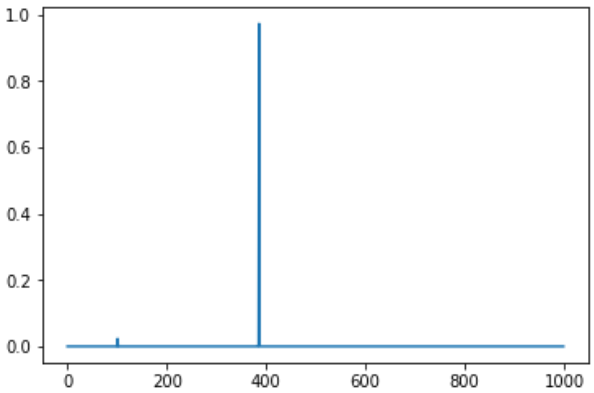
```
W0123 11:17:29.866466 140581987952384 deprecation_wrapper.py:119] From /usr/local/lib/python3.5/dist-packages/keras/backend/tensorflow_backend.py:422: The name tf.global_variables is deprecated. Please use tf.compat.v1.global_variables instead.
```

When we look at the dimensions of the output, we see that we have a vector of 1000 dimensions. Each dimensions corresponds to a category and the value represents the probability that the image contains that category. If we plot the vector we see that the image clearly belong to one category:

In [8]:
```
features.shape
```

Out[8]: (1, 1000)

In [9]:
```
plt.plot(features.T)
plt.show()
```




We can use the decond function, to know what this category index corresponds to:

In [10]:
```
decode_predictions(features, top=1000)
```

```
Downloading data from https://storage.googleapis.com/download.tensorflow.
org/data/imagenet_class_index.json
40960/35363 [==================================] - 0s 1us/step
```

```
Out[10]: [[('n02504458', 'African_elephant', 0.97247916),
           ('n01871265', 'tusker', 0.02319269),
           ('n02504013', 'Indian_elephant', 0.004200729),
           ('n02437312', 'Arabian_camel', 9.9511075e-05),
           ('n02100583', 'vizsla', 6.808777e-06),
           ('n02099849', 'Chesapeake_Bay_retriever', 2.5692e-06),
           ('n03124170', 'cowboy_hat', 1.1540138e-06),
           ('n01704323', 'triceratops', 1.0709498e-06),
           ('n02389026', 'sorrel', 1.0571564e-06),
           ('n02422106', 'hartebeest', 8.660311e-07),
           ('n02096051', 'Airedale', 7.121821e-07),
           ('n02090379', 'redbone', 6.9493774e-07),
           ('n02087394', 'Rhodesian_ridgeback', 6.583336e-07),
           ('n04604644', 'worm_fence', 5.9125585e-07),
           ('n02092339', 'Weimaraner', 5.736652e-07),
           ('n03124043', 'cowboy_boot', 5.6223433e-07),
           ('n01688243', 'frilled_lizard', 4.8969315e-07),
           ('n03697007', 'lumbermill', 3.873958e-07),
           ('n03404251', 'fur_coat', 3.7333308e-07),
           ('n04350905', 'suit', 3.6049698e-07),
           ('n04259630', 'sombrero', 3.3618875e-07),
           ('n04399382', 'teddy', 3.2949515e-07),
           ('n07734744', 'mushroom', 3.0491432e-07),
           ('n07754684', 'jackfruit', 2.5534945e-07),
           ('n02408429', 'water_buffalo', 2.43335e-07),
           ('n04458633', 'totem_pole', 2.3871908e-07),
           ('n04597913', 'wooden_spoon', 2.3287092e-07),
           ('n11879895', 'rapeseed', 2.2912964e-07),
           ('n02963159', 'cardigan', 2.2274801e-07),
           ('n07802026', 'hay', 1.8962464e-07),
           ('n02088466', 'bloodhound', 1.8776879e-07),
           ('n02129165', 'lion', 1.8023877e-07),
           ('n02410509', 'bison', 1.5872558e-07),
           ('n02403003', 'ox', 1.5586099e-07),
           ('n02454379', 'armadillo', 1.5301437e-07),
           ('n03498962', 'hatchet', 1.4770363e-07),
           ('n04208210', 'shovel', 1.4289928e-07),
           ('n01518878', 'ostrich', 1.2654296e-07),
           ('n02412080', 'ram', 1.2329042e-07),
           ('n02109047', 'Great_Dane', 1.1550219e-07),
           ('n04417672', 'thatch', 1.0752834e-07),
           ('n03134739', 'croquet_ball', 1.0582936e-07),
           ('n03000684', 'chain_saw', 1.0351832e-07),
           ('n02906734', 'broom', 9.7853764e-08),
           ('n04099969', 'rocking_chair', 9.2880164e-08),
           ('n04562935', 'water_tower', 9.1811906e-08),
           ('n02489166', 'proboscis_monkey', 9.11181e-08),
           ('n02793495', 'barn', 8.838817e-08),
           ('n04371430', 'swimming_trunks', 8.648289e-08),
           ('n02113799', 'standard_poodle', 8.531001e-08),
           ('n04599235', 'wool', 8.296619e-08),
           ('n02843684', 'birdhouse', 8.085067e-08),
           ('n03776460', 'mobile_home', 7.9733795e-08),
           ('n02012849', 'crane', 7.9576395e-08),
           ('n02099429', 'curly-coated_retriever', 7.6460026e-08),
           ('n02397096', 'warthog', 7.4261834e-08),
           ('n01677366', 'common_iguana', 7.244132e-08),
           ('n02391049', 'zebra', 6.915432e-08),
           ('n02095570', 'Lakeland_terrier', 6.5524716e-08),
           ('n02093991', 'Irish_terrier', 6.52822e-08),
           ('n03743016', 'megalith', 6.347313e-08),
           ('n04532670', 'viaduct', 6.3057904e-08),
           ('n02422699', 'impala', 5.915353e-08),
           ('n02093647', 'Bedlington_terrier', 5.7262092e-08),
           ('n02099601', 'golden_retriever', 5.716737e-08),
           ('n03803284', 'muzzle', 5.6893036e-08),
           ('n03873416', 'paddle', 5.4728215e-08),
           ('n01695060', 'Komodo_dragon', 5.4479095e-08),
```

The three best categories are three categories of different elephants, but the best one is indeed the African one.

## 16.4 Image with multiple content

What happens if multiple objects are in an image like here a dog and a cat or a banana and strawberries?

In [11]:
```
#image = skimage.io.imread('https://upload.wikimedia.org/wikipedia/commons/0/07/Chien-lit_%26_Chat-en-lit.jpg')
image = skimage.io.imread('https://live.staticflickr.com/3652/3295428010_9284075e7b_b.jpg')
```

In [12]:
```
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.show()
```


We preprocess the image and do the prediction:

In [13]:
```
model = VGG16(weights='imagenet', include_top=True)
#model = Xception(weights='imagenet', include_top=True)

image_resize = skimage.transform.resize(image,(224,224),preserve_range=True)
x = np.expand_dims(image_resize, axis=0)
x = preprocess_input(x)

features = model.predict(x)
```

```
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:105: UserWarning: The default mode, 'constant', will be changed to 'reflect' in skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:110: UserWarning: Anti-aliasing will be enabled by default in skimage 0.15 to avoid aliasing artifacts when down-sampling images.
  warn("Anti-aliasing will be enabled by default in skimage 0.15 to "
```

In [14]:
```
decode_predictions(features, top=1000)
```

```
Out[14]: [[('n07753592', 'banana', 0.5761249),
           ('n07745940', 'strawberry', 0.19425765),
           ('n07753275', 'pineapple', 0.05217132),
           ('n07614500', 'ice_cream', 0.030278875),
           ('n07749582', 'lemon', 0.015831826),
           ('n07760859', 'custard_apple', 0.012895143),
           ('n07753113', 'fig', 0.011580526),
           ('n07747607', 'orange', 0.009989414),
           ('n04476259', 'tray', 0.009962709),
           ('n07579787', 'plate', 0.0068863747),
           ('n07718472', 'cucumber', 0.006387197),
           ('n04332243', 'strainer', 0.0054645333),
           ('n07768694', 'pomegranate', 0.0054520713),
           ('n07836838', 'chocolate_sauce', 0.003862604),
           ('n07742313', 'Granny_Smith', 0.0028679618),
           ('n04597913', 'wooden_spoon', 0.0027667894),
           ('n03461385', 'grocery_store', 0.0026932321),
           ('n07716358', 'zucchini', 0.0026920456),
           ('n07613480', 'trifle', 0.0022338615),
           ('n07583066', 'guacamole', 0.00217574),
           ('n03089624', 'confectionery', 0.0013857629),
           ('n07932039', 'eggnog', 0.0013704551),
           ('n04204238', 'shopping_basket', 0.0013690287),
           ('n07717556', 'butternut_squash', 0.0013453267),
           ('n07718747', 'artichoke', 0.0012477095),
           ('n07930864', 'cup', 0.0011815256),
           ('n02909870', 'bucket', 0.0010817345),
           ('n03775546', 'mixing_bowl', 0.0010295234),
           ('n03944341', 'pinwheel', 0.0009666784),
           ('n03127925', 'crate', 0.0009592887),
           ('n07714990', 'broccoli', 0.00087834854),
           ('n03729826', 'matchstick', 0.0007504259),
           ('n02776631', 'bakery', 0.00071163604),
           ('n02971356', 'carton', 0.0006731103),
           ('n03482405', 'hamper', 0.00065947045),
           ('n07720875', 'bell_pepper', 0.00064582424),
           ('n03633091', 'ladle', 0.0006443229),
           ('n07892512', 'red_wine', 0.00063594204),
           ('n03445777', 'golf_ball', 0.00061149464),
           ('n03786901', 'mortar', 0.0006095598),
           ('n03908618', 'pencil_box', 0.00054616673),
           ('n03720891', 'maraca', 0.00052341406),
           ('n04399382', 'teddy', 0.0005185749),
           ('n12620546', 'hip', 0.00051782426),
           ('n07715103', 'cauliflower', 0.00050635735),
           ('n07871810', 'meat_loaf', 0.00050255464),
           ('n03047690', 'clog', 0.0004752425),
           ('n07693725', 'bagel', 0.0004202755),
           ('n07716906', 'spaghetti_squash', 0.000415875),
           ('n01945685', 'slug', 0.00041408345),
           ('n01734418', 'king_snake', 0.0004139101),
           ('n04270147', 'spatula', 0.0004090329),
           ('n03950228', 'pitcher', 0.00040275132),
           ('n07717410', 'acorn_squash', 0.00039611929),
           ('n02110341', 'dalmatian', 0.00039234685),
           ('n04263257', 'soup_bowl', 0.00039161675),
           ('n04259630', 'sombrero', 0.00038066693),
           ('n03991062', 'pot', 0.00036915473),
           ('n04133789', 'sandal', 0.00030882948),
           ('n07880968', 'burrito', 0.00030034475),
           ('n04141975', 'scale', 0.00029977187),
           ('n07734744', 'mushroom', 0.0002680286),
           ('n03133878', 'Crock_Pot', 0.00026488805),
           ('n04026417', 'purse', 0.0002394798),
           ('n03041632', 'cleaver', 0.00022614613),
           ('n03063599', 'coffee_mug', 0.00022026774),
           ('n02526121', 'eel', 0.00021577944),
           ('n04317175', 'stethoscope', 0.00021190982),
```

We end up with probabilitis split among multiple categories of a certain "style" like multiple dog breeds. One way to try improving on this, is to use this classifier to do an approximative segmentation by splitting the image into subregions.

We create overlapping patches and do the prediction on those:

In [15]:
```
patch = 400
step = 100
all_features =[]
for i in np.arange(0,image.shape[0]-patch-1,step):
    print(i)
    for j in np.arange(0,image.shape[1]-patch-1,step):
        subimage = image[i:i+patch,j:j+patch,:]
        image_resize = skimage.transform.resize(subimage,(224,224),prese
rve_range=True)
        x = np.expand_dims(image_resize, axis=0)
        x = preprocess_input(x)

        features = model.predict(x)
        all_features.append(features)
```

```
0
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:105: U
serWarning: The default mode, 'constant', will be changed to 'reflect' in
skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:110: U
serWarning: Anti-aliasing will be enabled by default in skimage 0.15 to a
void aliasing artifacts when down-sampling images.
  warn("Anti-aliasing will be enabled by default in skimage 0.15 to "

100
200
```

In [16]:
```
[decode_predictions(x, top=1000)[0][0] for x in all_features if decode_p
redictions(x, top=1000)[0][0][2]>0.3]
```

Out[16]:
```
[('n07753592', 'banana', 0.78769964),
 ('n07753592', 'banana', 0.47260636),
 ('n07753592', 'banana', 0.5114516),
 ('n07745940', 'strawberry', 0.68358153),
 ('n07745940', 'strawberry', 0.81029093),
 ('n07745940', 'strawberry', 0.92421764),
 ('n07753592', 'banana', 0.8903582),
 ('n07753592', 'banana', 0.7702213),
 ('n07753592', 'banana', 0.8909377),
 ('n07745940', 'strawberry', 0.9785351),
 ('n07745940', 'strawberry', 0.9900946),
 ('n07745940', 'strawberry', 0.98152024),
 ('n07745940', 'strawberry', 0.7820029),
 ('n07753592', 'banana', 0.98062783),
 ('n07753592', 'banana', 0.80547625),
 ('n07745940', 'strawberry', 0.69026893),
 ('n07745940', 'strawberry', 0.99909794),
 ('n07745940', 'strawberry', 0.99729496),
 ('n07745940', 'strawberry', 0.9918213),
 ('n07745940', 'strawberry', 0.4741534)]
```

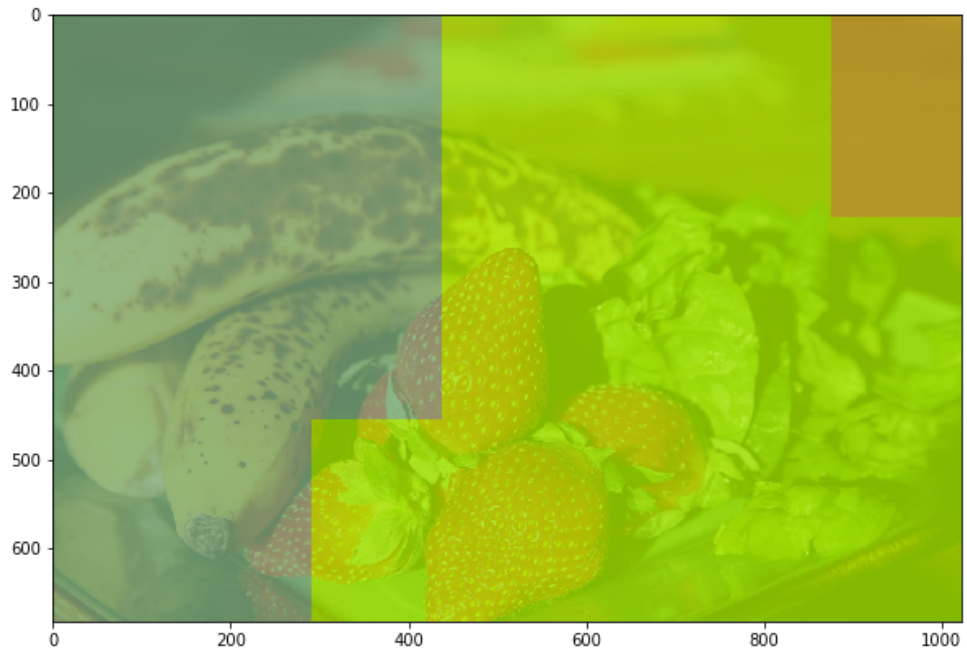
We can now superpose those segmented features over the original image:

In [17]:
```
import matplotlib.colors
cmap = matplotlib.colors.ListedColormap ( np.random.rand ( 256,3))

reshaped = np.reshape([np.argmax(x) for x in all_features],
                      (len(np.arange(0,image.shape[0]-patch-1,step)),len
(np.arange(0,image.shape[1]-patch-1,step))))

plt.figure(figsize=(10,10))
plt.imshow(image)
plt.imshow(skimage.transform.resize(reshaped,(image.shape[0],image.shape
[1]), order=0,preserve_range=True),cmap = cmap,alpha = 0.8)
plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:105: U
serWarning: The default mode, 'constant', will be changed to 'reflect' in
skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:110: U
serWarning: Anti-aliasing will be enabled by default in skimage 0.15 to a
void aliasing artifacts when down-sampling images.
  warn("Anti-aliasing will be enabled by default in skimage 0.15 to "
```




Let's create an array with the index names and plot them on top of the image:

In [18]:
```
names = np.reshape([decode_predictions(x, top=1000)[0][0][1] for x in al
l_features],
          (len(np.arange(0,image.shape[0]-patch-1,step)),len(np.arange
(0,image.shape[1]-patch-1,step))))
```

In [19]:
```
plt.figure(figsize=(10,10))
plt.imshow(image)
plt.imshow(skimage.transform.resize(reshaped,(image.shape[0],image.shape
[1]), order=0,preserve_range=True),cmap = cmap,alpha = 0.8)
fact = image.shape[0]/reshaped.shape[0]
for x in range(names.shape[0]):
    for y in range(names.shape[1]):
        plt.text(x=(y)*image.shape[1]/reshaped.shape[1],y=(x+0.5)*image.
shape[0]/reshaped.shape[0],s = names[x,y])
plt.show()
```

```
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:105: U
serWarning: The default mode, 'constant', will be changed to 'reflect' in
skimage 0.15.
  warn("The default mode, 'constant', will be changed to 'reflect' in "
/usr/local/lib/python3.5/dist-packages/skimage/transform/_warps.py:110: U
serWarning: Anti-aliasing will be enabled by default in skimage 0.15 to a
void aliasing artifacts when down-sampling images.
  warn("Anti-aliasing will be enabled by default in skimage 0.15 to "
```