ImageClassification

February 28, 2020

1 Classification on galaxy zoo dataset

https://www.kaggle.com/c/galaxy-zoo-the-galaxy-challenge/overview

1.0.1 Data includes images with unique IDs (jpg files) and a table containing the probabilities of each class in the columns

1.0.2 Read table with classes

[2]:
```
import matplotlib as mpl
%matplotlib inline
import matplotlib.pyplot as plt
import numpy as np
import os
import tarfile
import urllib
import pandas as pd
import seaborn as sns
from glob import glob
from tqdm import tqdm

# Use pandas to retrieve data file
#url = '' # use to fetch data file each time

# Read in CSV file with probabilities
df = pd.read_csv("training_solutions_rev1.csv",index_col=0)
#classprobs.values.astype(np.float32)
df.info()
```

```
<class 'pandas.core.frame.DataFrame'>
Int64Index: 61578 entries, 100008 to 999967
Data columns (total 37 columns):
 #   Column     Non-Null Count  Dtype
---  ------     --------------  -----
 0   Class1.1   61578 non-null  float64
 1   Class1.2   61578 non-null  float64
 2   Class1.3   61578 non-null  float64
```

```
 3   Class2.1   61578 non-null  float64
 4   Class2.2   61578 non-null  float64
 5   Class3.1   61578 non-null  float64
 6   Class3.2   61578 non-null  float64
 7   Class4.1   61578 non-null  float64
 8   Class4.2   61578 non-null  float64
 9   Class5.1   61578 non-null  float64
 10  Class5.2   61578 non-null  float64
 11  Class5.3   61578 non-null  float64
 12  Class5.4   61578 non-null  float64
 13  Class6.1   61578 non-null  float64
 14  Class6.2   61578 non-null  float64
 15  Class7.1   61578 non-null  float64
 16  Class7.2   61578 non-null  float64
 17  Class7.3   61578 non-null  float64
 18  Class8.1   61578 non-null  float64
 19  Class8.2   61578 non-null  float64
 20  Class8.3   61578 non-null  float64
 21  Class8.4   61578 non-null  float64
 22  Class8.5   61578 non-null  float64
 23  Class8.6   61578 non-null  float64
 24  Class8.7   61578 non-null  float64
 25  Class9.1   61578 non-null  float64
 26  Class9.2   61578 non-null  float64
 27  Class9.3   61578 non-null  float64
 28  Class10.1  61578 non-null  float64
 29  Class10.2  61578 non-null  float64
 30  Class10.3  61578 non-null  float64
 31  Class11.1  61578 non-null  float64
 32  Class11.2  61578 non-null  float64
 33  Class11.3  61578 non-null  float64
 34  Class11.4  61578 non-null  float64
 35  Class11.5  61578 non-null  float64
 36  Class11.6  61578 non-null  float64
dtypes: float64(37)
memory usage: 17.9 MB
```

1.1 Read images

1.1.1 Using PIL: define read function to crop/resize image but doens't load pixel data to conserve memory

1.1.2 `np.vectorize` makes read function work with iterables like a list of filenames

[229]:
```
# Image reading function that conserves memory
from PIL import Image

def read_image(file,crop_size=(200,200),out_size=(64,64)):
    with Image.open(file) as im:
        return im.convert('L').crop(((im.size[1]-crop_size[1])//2, (im.
 ↪size[0]-crop_size[0])//2, \
                (im.size[1]+crop_size[1])//2, (im.size[0]+crop_size[0])//2 )).
 ↪resize(out_size)

def load_image(PILimage):
    return Image.load(PILImage,'F').asarray()

read_image_list = np.
 ↪vectorize(read_image,excluded=['crop_size','out_size'],otypes=np.
 ↪array([Image.Image]))

load_image_list = np.vectorize(load_image,otypes=[np.ndarray])
#load_image_data = np.vectorize(Image.
 ↪load,excluded=['crop_size','out_size'],otypes=np.array([np.ndarray]))
```

Use `glob` + `np.sort` to define a sorted array of image filenames

Define crop and resize parameters as keyword args

[230]:
```
path='images_training_rev1/*'
flist = np.sort(np.array(glob(path)))
```

Call `read_image_list` on subset of images for plotting and data exploration

[231]:
```
kwargs={'crop_size':crop_size,'out_size':out_size}
imlist_sub=read_image_list(flist[:100])
im_sub = load_image_list(imlist_sub)
im_sub
```

```
␣
↪---------------------------------------------------------------------------

AttributeError                            Traceback (most recent call␣
↪last)
```

```
        <ipython-input-231-714cad07e5cf> in <module>
          1 kwargs={'crop_size':crop_size,'out_size':out_size}
          2 imlist_sub=read_image_list(flist[:100])
    ----> 3 im_sub = load_image_list(imlist_sub)
          4 im_sub


        ~/opt/anaconda3/envs/mlpy/lib/python3.7/site-packages/numpy/lib/
↪function_base.py in __call__(self, *args, **kwargs)
       2089             vargs.extend([kwargs[_n] for _n in names])
       2090
    -> 2091         return self._vectorize_call(func=func, args=vargs)
       2092
       2093     def _get_ufunc_and_otypes(self, func, args):


        ~/opt/anaconda3/envs/mlpy/lib/python3.7/site-packages/numpy/lib/
↪function_base.py in _vectorize_call(self, func, args)
       2165                         for a in args]
       2166
    -> 2167             outputs = ufunc(*inputs)
       2168
       2169             if ufunc.nout == 1:


        <ipython-input-229-406eb17c2d29> in load_image(PILimage)
          8
          9 def load_image(PILimage):
    ---> 10     return Image.load(PILImage,'F').asarray()
         11
         12 read_image_list = np.
↪vectorize(read_image,excluded=['crop_size','out_size'],otypes=np.array([Image.
↪Image]))


        AttributeError: module 'PIL.Image' has no attribute 'load'
```

[214]:

```
[214]: array([<PIL.Image.Image image mode=L size=64x64 at 0x11DF3A390>,
        <PIL.Image.Image image mode=L size=64x64 at 0x12CB61A90>,
        <PIL.Image.Image image mode=L size=64x64 at 0x12B2ED910>,
        <PIL.Image.Image image mode=L size=64x64 at 0x12B2ED990>,
        <PIL.Image.Image image mode=L size=64x64 at 0x12B2EDC50>,
        <PIL.Image.Image image mode=L size=64x64 at 0x11DF3AF50>,
```

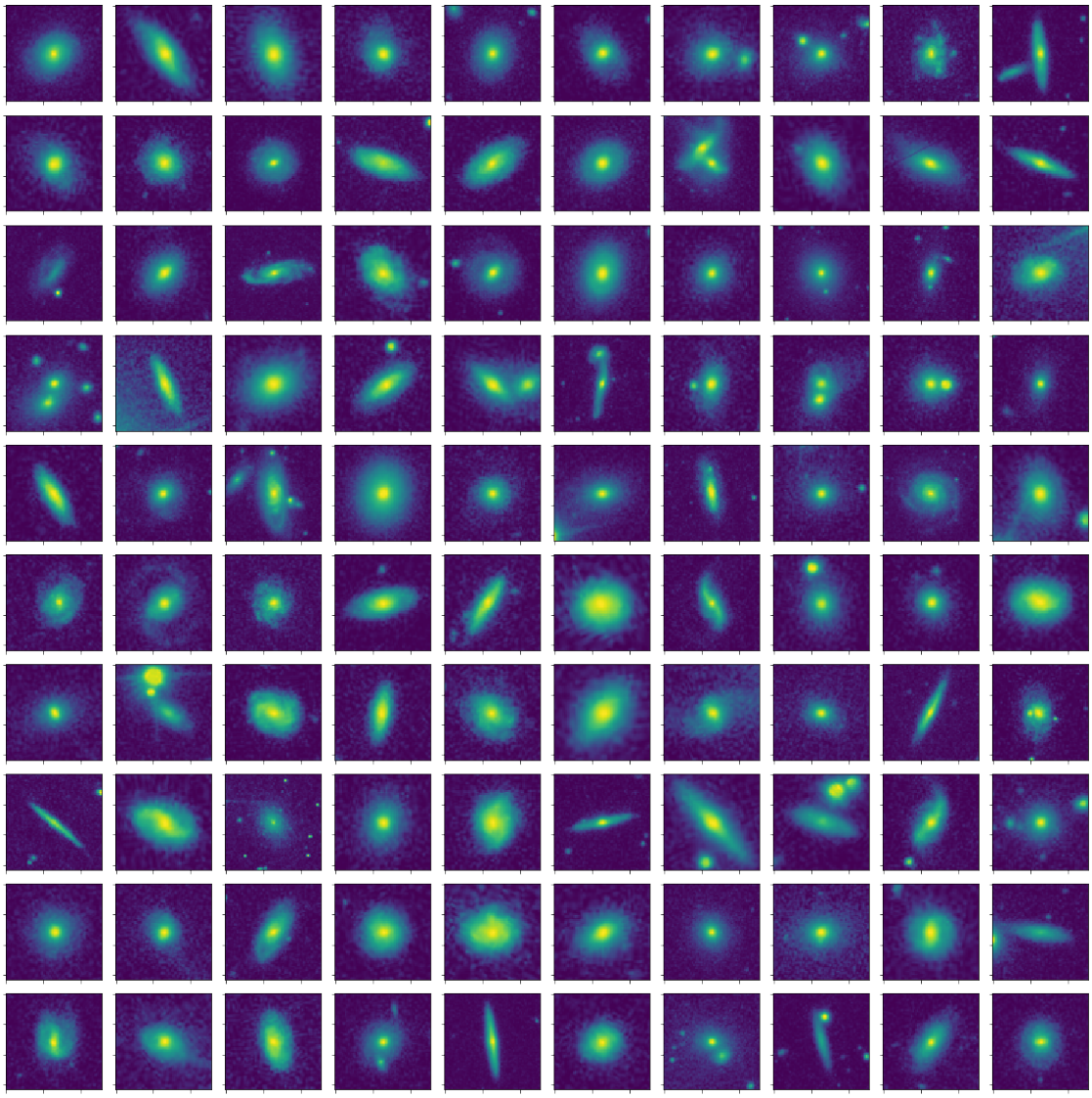
<PIL.Image.Image image mode=L size=64x64 at 0x11DF3A350>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EDCD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA850>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA510>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EAAD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA490>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA590>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA290>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA690>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA110>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EAED0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12913B050>,
<PIL.Image.Image image mode=L size=64x64 at 0x12913B090>,
<PIL.Image.Image image mode=L size=64x64 at 0x11DF3A2D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2E3FD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2E3950>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA650>,
<PIL.Image.Image image mode=L size=64x64 at 0x129149F10>,
<PIL.Image.Image image mode=L size=64x64 at 0x12A3DCD50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA5D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x11F3ACF90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12C557050>,
<PIL.Image.Image image mode=L size=64x64 at 0x12C557210>,
<PIL.Image.Image image mode=L size=64x64 at 0x129149FD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2EA350>,
<PIL.Image.Image image mode=L size=64x64 at 0x12C557310>,
<PIL.Image.Image image mode=L size=64x64 at 0x129149DD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F05D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0D50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0490>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0750>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F04D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0590>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0E50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0610>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0A50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0D10>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F09D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0CD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0390>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0410>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0D90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0650>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0890>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0A90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0ED0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0690>,

<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0950>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0990>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0450>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0C50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F03D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0F50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0190>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F02D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0050>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0110>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F01D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0090>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0210>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0BD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0DD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F0710>,
<PIL.Image.Image image mode=L size=64x64 at 0x12C5570D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F11D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1C50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1450>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1D10>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1390>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1A90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1510>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1F50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1D50>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F14D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1150>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1FD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1F90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F19D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1B10>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1AD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1ED0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1E10>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1A10>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1850>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1750>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1B90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1890>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1310>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F17D0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1910>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1C90>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1BD0>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1990>,
<PIL.Image.Image image mode=L size=64x64 at 0x12B2F1DD0>],

```
      dtype=object)
```

[195]:
```
import skimage.io
from skimage.io import imshow_collection

#help(imshow_collection)
#fig,axs=plt.subplots(10,10)
fig=plt.figure(figsize=(16,16))
gs=fig.add_gridspec(nrows=10,ncols=10)
k=0
for i in range(10):
    for j in range(10):
        ax_ij = fig.add_subplot(gs[i,j])
        ax_ij.imshow(imlist_sub[k])
        ax_ij.set_xticklabels([])
        ax_ij.set_yticklabels([])
        k+=1
fig.tight_layout()
#skimage.io.show()
plt.show()
```



1.2 Now do batch learning: create binary classifier for Q1.1: Is is smooth/round or is it not?

[191]:
```
df_c1 = df.loc[:,"Class1.1":"Class1.3"].copy()

Y_train = (df_c1.values.argmax(axis=1) == 0) # now target is simply ROUND/
 ↪SMOOTH OR NOT

Y_train
```

[191]:
```
array([False, False,  True, …, False, False,  True])
```

[192]:
```
batch_size = 5000

N_samp = df.shape[0]

print("Implementing %i batches with %i samples each"%(N_batches,batch_size))
```

```
Implementing 13 batches with 5000 samples each
```

[206]:
```
from sklearn.linear_model import SGDClassifier
from sklearn import metrics
from sklearn.model_selection import train_test_split

# define image processing parameters
s_crop=(200,200)
s_image=(64,64)
kwargs={'crop_size':s_crop,'out_size':s_image}


# initialize the classifier
sgd_clf = SGDClassifier(random_state=1, loss='modified_huber', penalty='l2',␣
 ↪tol=1.0e-3, \
                        verbose=1, shuffle=False, warm_start=False,␣
 ↪early_stopping=False)

n_batch=[]
accuracy=[]
n=0
while n*batch_size < N_samp: # save last two batches for testing
    # define X of batch (reshape images if necessary)
    #print("INDEX %i : %i"%(n * batch_size, (n+1)*batch_size))

#     if (n==N_batches-1): # if the last batch
#         X_train_batch = read_image_list( flist[n * batch_size : ] )#.
 ↪concatenate().\
#         reshape((np.size(imcl.files[n *batch_size:]),-1))
#         Y_train_batch = Y_train_bin1[n * batch_size : ]
 #   else:
    X_batch = read_image_list(flist[n * batch_size :␣
 ↪(n+1)*batch_size],**kwargs) #.concatenate().reshape((batch_size,-1))
    Y_batch = Y_train[n * batch_size : (n+1)*batch_size]


   ␣
 ↪X_train_batch,X_test_batch,Y_train_batch,Y_test_batch=train_test_split(X_batch,Y_batch,rand

    print(X_train_batch,Y_train_batch)

    n+=1
```

```
    break
    # FIT
 #   sgd_clf.partial_fit(X_train_batch, Y_train_batch, classes=[True,False])

#     # ACCURACY
#     batch_score = sgd_clf.score(X_train_batch, Y_train_batch)

#     print("Accuracy score (train): %.1f PRECENT"%(batch_score*100))
#     n_batch.append(n)
#     accuracy_score.append(batch_score)
```

```
      ␣
↪---------------------------------------------------------------------------

       AttributeError                            Traceback (most recent call␣
↪last)

       <ipython-input-206-276dc6d43eec> in <module>
        25 #         Y_train_batch = Y_train_bin1[n * batch_size : ]
        26  #   else:
   ---> 27     X_batch = read_image_list(flist[n * batch_size :␣
↪(n+1)*batch_size],**kwargs).getdata() #.concatenate().reshape((batch_size,-1))
        28     Y_batch = Y_train[n * batch_size : (n+1)*batch_size]
        29


       AttributeError: 'numpy.ndarray' object has no attribute 'getdata'
```

[ ]:

1.2.1 Using skimage: ImageCollection

[29]:
```
# # display image
# from skimage import data,filters,io
# from skimage import img_as_float32, img_as_uint, img_as_ubyte
# from sklearn.base import BaseEstimator, TransformerMixin
# from skimage.feature import hog
# from skimage.io import imread,imshow_collection,ImageCollection
# from skimage.transform import rescale,resize
# # from skimage.io import ImageCollection, concatenate_images
# # from skimage.util import img_as_ubyte,img_as_uint,img_as_float

# # img_shape = (424,424)
# # crop_shape = (256,256)
```

```
# # train_img_shape = (32,32)

# #flist = np.sort(np.array(glob('images_training_rev1/*')))
# flist = ['images_training_rev1/%i.jpg'%i for i in Y_train_class1.index[:]]

# #~~~ DEFINE FUNCTION TO READ IN IMAGES AND CROP, FLATTEN

# # crop image (np array) from original size to output sieze
# def crop(image,size_orig,size_crop):
#     return image[(size_orig-size_crop)//2:(size_orig+size_crop)//2,\
#                  (size_orig-size_crop)//2:(size_orig+size_crop)//2]

# # read image file and crop
# def read_crop(f,size_orig,size_crop,size_image):
#     i=crop(plt.imread(f), size_orig,size_crop)
#     i=resize(i,(size_image,size_image))
#     return i.astype(np.float32)
```

[30]:
```
img_dim = 424
crop_dim= 192
resize_dim = 64

#~~ DEFINE IMAGE COLLECTION

load_func_kwargs={'size_orig':img_dim,'size_crop':crop_dim, 'size_image':
 ↪resize_dim}
#print(flist)
imcl = ImageCollection(flist, load_func=read_crop, conserve_memory=True,␣
 ↪**load_func_kwargs)
```

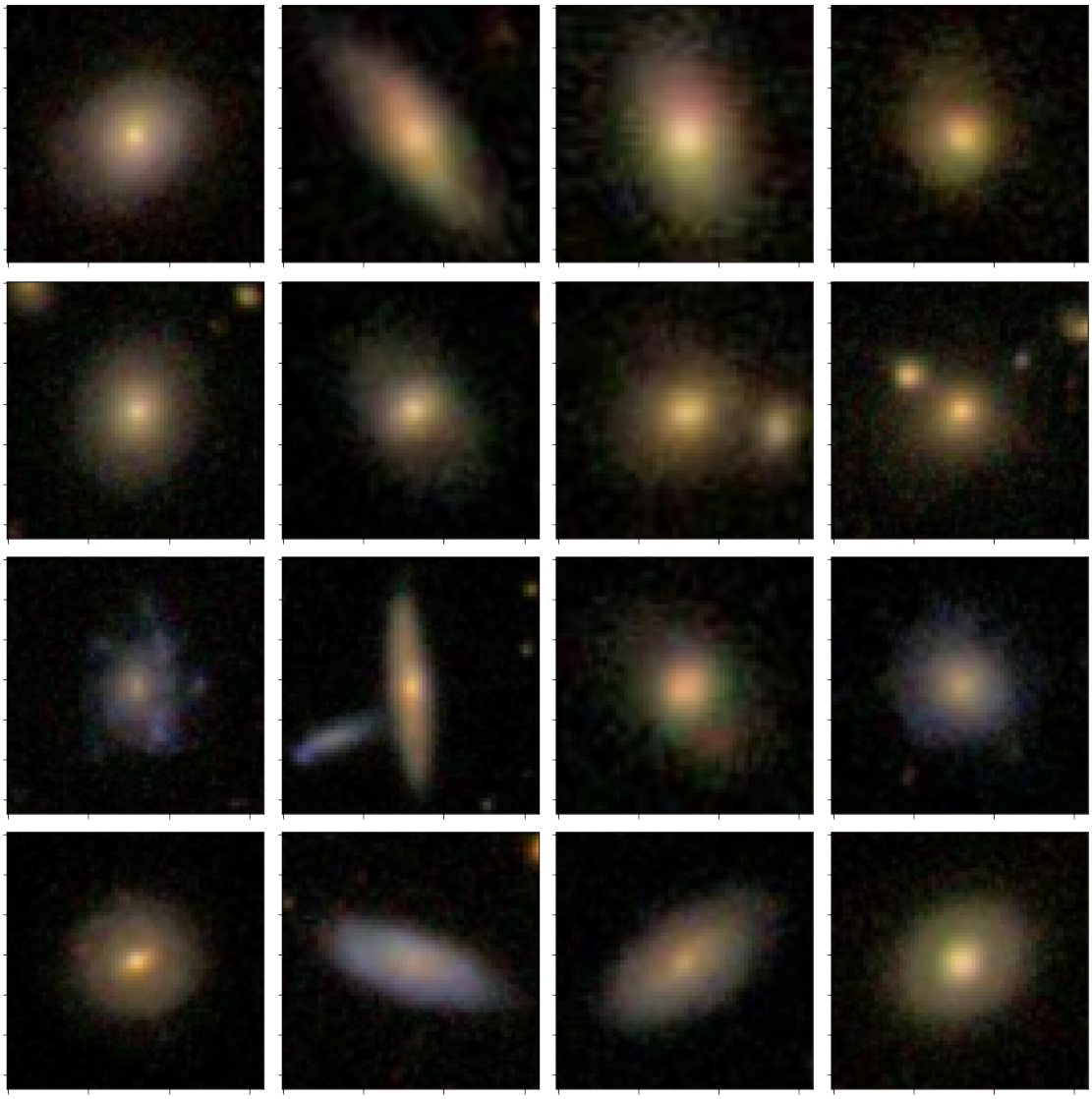
[31]:
```
imcl_plot = ImageCollection(flist[:16], load_func=read_crop,␣
 ↪conserve_memory=False,**load_func_kwargs)

%matplotlib inline

#img_subset=ic[:16]
#print(img_subset[0].shape)
#img_subset=ic.concatenate()#[:16])
#print(img_subset.shape)
#plt.figure(figsize=(16,16))
#plt.figure(figsize=(16,16))
#plt.figure(figsize=(20,20))

ax_im=io.imshow_collection(imcl_plot)
ax_im.set_size_inches(12,12)
for ax in ax_im.axes:
    ax.set_xticklabels([])
```

```
    ax.set_yticklabels([])
ax_im.tight_layout()
io.show()
#ax_im.
```



1.3 Make train set, X_train will be list of images. Use batch learning to save memory.

1.3.1 Prepare data: flatten images (X_train) and create binary classifier for Q1.1: Is is smooth/round?

1.3.2 Trying SGDClassifier first

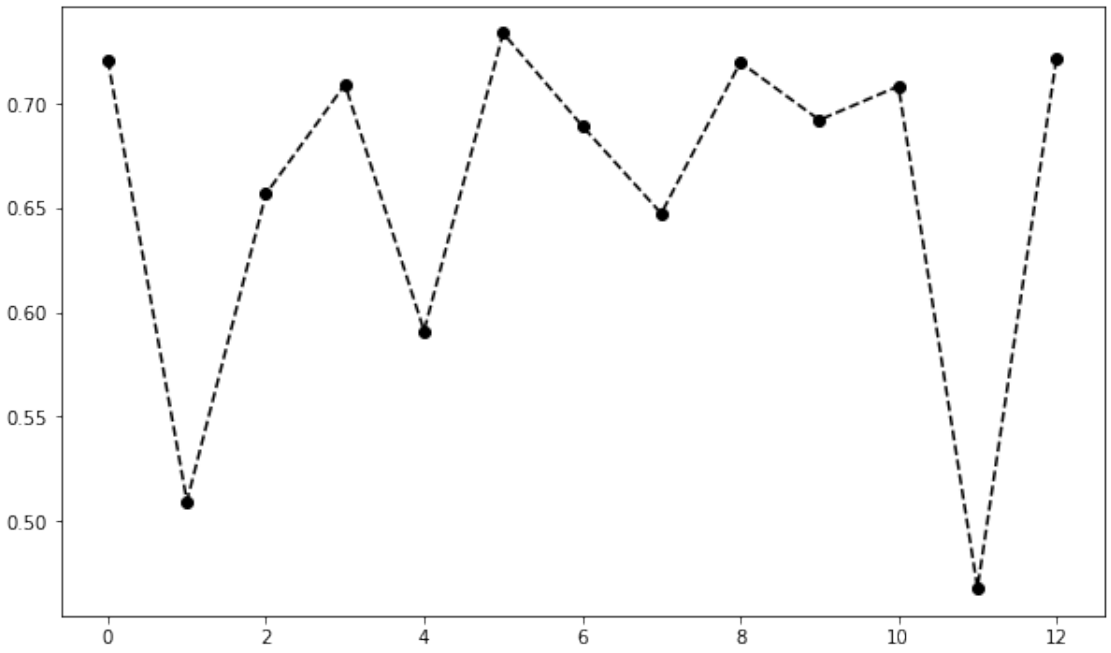
[55]:
```
from sklearn.linear_model import SGDClassifier

sgd_clf = SGDClassifier(loss='hinge', penalty='l2', n_jobs=3, tol=1.0e-5,␣
↪alpha=1.0e-4, \
                        verbose=1, shuffle=False, warm_start=False,␣
↪early_stopping=False)
```

1.3.3 Loop through batches, learn

[57]:
```
fig=plt.figure(figsize=(10,6))
plt.plot(n_batch,accuracy_score,'ko--')
```

[57]:
```
[<matplotlib.lines.Line2D at 0x116101fd0>]
```



2 nonsense below

[ ]:
```
sgd_clf.partial_fit(X_train_b1, Y_train_b1, classes=[True,False])
```

[ ]:
```
accuracy=sgd_clf.score(X_train_b1,Y_train_b1)
print("Accuracy of batch 1: ",accuracy)
```

[ ]:
```
X_train_b2 = imcl[batch_size:2*batch_size].concatenate().reshape( (5000, -1))
Y_train_b2 = Y_train_bin1[batch_size:2*batch_size]
```

[ ]:
```
sgd_clf.partial_fit(X_train_b2, Y_train_b2)
```

[ ]:
```
accuracy2=sgd_clf.score(X_train_b2,Y_train_b2)
print("Accuracy including batch 2: ",accuracy2)
```

[ ]:
```
X_train_
```

[ ]:
```

```

[ ]:
```
knn_clf.fit(X_train_prep,Y_train_prep)
```

[ ]:
```
help(knn_clf.score)
#score=knn_clf.score(X_train_prep,Y_train_prep)
```

[ ]:
```
print(score)
```

[ ]:
```
X_test_prep = scaler.transform(X_test)
Y_test_binary = (Y_test >= 0.5)
Y_test_prep = binarizer.transform(Y_test_binary)
```

[ ]:
```
print(X_test_prep[0].shape)
```

[ ]:
```
Y_test_pred = knn_clf.predict([X_test_prep[0]])
```

[ ]:
```
print(Y_test_pred[0,:])
```

[ ]:
```
plt.figure(figsize=(12,12))
#print(Y.index[8000])
#print(probs.columns[1:].values)
Y_test_true = Y.iloc[8000,:]
#u,w=zip((Y.iloc[8000,:]).values,(Y.iloc[8000,:]).keys())#[1:3])

im_0 = mpl.image.imread("images_training_rev1/%s.jpg"%Y.index[8000])
plt.imshow(im_0,origin='lower')
for j in np.arange(0,Y_test.shape[1]):
    # print(j)
```

```
    cj=probs.columns[j]
  #  print(cj,j)
    #print(Y[c])#columns[1:])
   # print(j,c)
    print("%s "%j,"%s"%cj, "  TRUE = %s "%Y_test[0,j],\
          "PRED = %s"%Y_test_pred[0,j])#,\
                      # Y_test_pred[1+j]))
```

[ ]:
```
from sklearn.metrics import classification_report

print(classification_report(Y_test_prep[0], Y_test_pred))
```

[ ]:
```

```

2.0.1 QUESTION: how do you predict probability of classification?

2.0.2 Evaluate classifier using the test set

[ ]:
```
y_test_pred = clf.predict(X_test_scaled)
y_test_scores = clf.decision_function(X_test_scaled)
y_test_score = clf.score( X_test_scaled, y_test)

Cij_test = confusion_matrix( y_test, y_test_pred)
cv_test_scores = cross_val_score(clf, X_test_scaled, y_test, cv=3,␣
 ↪scoring='accuracy')

print(Cij_test)
print(cv_test_scores)
print(" ".join(Cij_test.ravel().astype('str')))
```

2.0.3 TEST SET RESULTS:

CONFUSION MATRIX:

[ ]:
```
from IPython.display import display, Math
display(Math(r'%i %i \\ %i %i'.
 ↪format(Cij_test)))#(Cij_test[0,0],Cij_test[0,1],Cij_test[1,0],Cij_test[1,1])))
```

[ ]:
```

```