Cats & Dogs

May 15, 2021

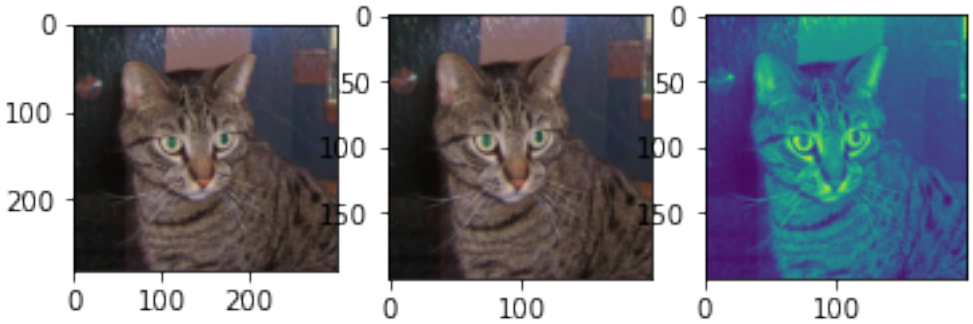
1 IRGUI ILYAS

1.1 TP Cats & Dogs

1.1.1 Le programme intégré dans le support de cours

[8]:
```
import numpy as np
import matplotlib.pyplot as plt
from skimage import io
from skimage.transform import resize
from skimage.color import rgb2gray
```

[9]:
```
#Exemple de traitement d'un sample du dataset avec les méthodes de scikit-image␣
↪(skimage)
cat1 = io.imread("cat.1.jpg")
cat1_ = resize(cat1, (200,200,3))
cat1_gray = rgb2gray(cat1_)
fig = plt.figure()
columns = 3; rows = 1
fig.add_subplot(rows, columns, 1);plt.imshow(cat1)
fig.add_subplot(rows, columns, 2);plt.imshow(cat1_)
fig.add_subplot(rows, columns, 3);plt.imshow(cat1_gray)
plt.show()
```



[10]:
```
x_train = []; y_train = []
for i in range(1,2001):
    cat = rgb2gray(resize(io.imread('training_set\cats\cat.{}.jpg'.format(i)),␣
 ↪(200,200)))#rendre les images d'entrainement en niveau de gris après␣
 ↪recadrage
    x_train.append(cat); y_train.append(0) #0-->'cat'

for i in range(1,2001):
    dog = rgb2gray(resize(io.imread('training_set\dogs\dog.{}.jpg'.format(i)),␣
 ↪(200,200)))
    x_train.append(dog); y_train.append(1) #1-->'dog'

x_train, y_train = np.asarray(x_train), np.asarray(y_train)
print('x_train shape: ',x_train.shape, 'y_train shape: ', y_train.shape)
```

```
x_train shape:  (4000, 200, 200) y_train shape:  (4000,)
```

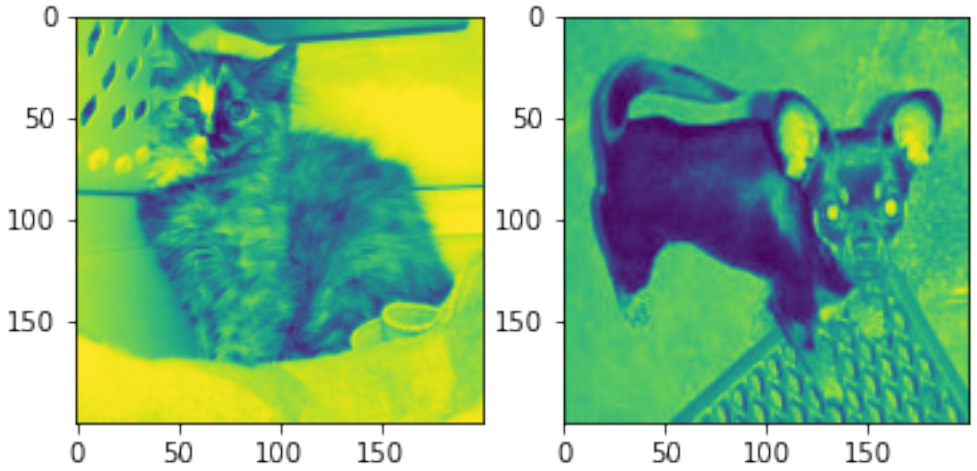
[11]:
```
x_test = []; y_test = []
for i in range(4001,5001):
    cat = rgb2gray(resize(io.imread('test_set\cats\cat.{}.jpg'.format(i)),␣
 ↪(200,200)))#rendre les images de test en niveau de gris après recadrage
    x_test.append(cat); y_test.append(0) #0-->'cat'

for i in range(4001,5001):
    dog = rgb2gray(resize(io.imread('test_set\dogs\dog.{}.jpg'.format(i)),␣
 ↪(200,200)))
    x_test.append(dog); y_test.append(1) #1-->'dog'

x_test, y_test = np.asarray(x_test), np.asarray(y_test)
print('x_test shape: ',x_test.shape, 'y_test shape: ', y_test.shape)
```

```
x_test shape:  (2000, 200, 200) y_test shape:  (2000,)
```

[12]:
```
def predict(X, k):
    distances = []
    for i in range(0, len(x_train)):
        distances += [np.sum(np.abs(x_train[i] - X))]
    min_indexs = np.argsort(distances)[:k]#récupérer les indices des valeurs␣
 ↪triés dans l'ordre croisant
    y_ = y_train[min_indexs]; counts = np.bincount(y_)#compter les occurances␣
 ↪de la valeur '0' et '1'
    if np.argmax(counts)==0:return('cat')#argmax(count) pour compter la valeur␣
 ↪qui se répète le plus (0 ou 1)
    else:return('dog')
```

[13]:
```
#Exemple de prediction en utilisant la fonction 'predict'
numeros_images_a_predire = [4,1089]
fig=plt.figure(); predictions=[]
```

```
columns=2;rows=1;i=1
for num in numeros_images_a_predire:
    predictions+=[predict(x_test[num],3)]
    fig.add_subplot(rows, columns, i); plt.imshow(x_test[num]); i+=1
plt.show()
print(predictions)
```




```
['cat', 'dog']
```

1.1.2 Le modèle basé sur KNN pour classifier les cats and dog

[14]:
```
#print(x_train.shape)..."x_train est un tableau 3D qui doit être converti en␣
 ↪tableau 2d via la fonction reshape"
x_train = x_train.reshape(x_train.shape[0], -1)
x_test = x_test.reshape(x_test.shape[0], -1)

from sklearn.neighbors import KNeighborsClassifier

neigh = KNeighborsClassifier(n_jobs=-1)#n_jobs permet d'utiliser tous les cores␣
 ↪disponibles du processeur.
neigh.fit(x_train,y_train)

score_train = neigh.score(x_train,y_train) #la performance en utilisant la base␣
 ↪d'apprentissage
print("Le score sur training set : {:.2f}%".format(score_train*100))

score_test = neigh.score(x_test,y_test) #la performance en utilisant la base de␣
 ↪test
print("Le score sur test set : {:.2f}%".format(score_test*100))
```

```
Le score sur training set : 70.75%
Le score sur test set : 54.55%
```

1.1.3 Optimisation des performances de KNN

En utilisant RandomizedSearchCV

[29]:
```
from sklearn.model_selection import RandomizedSearchCV
import time

distances=['euclidean','cityblock'];valeurs_de_k=np.arange(1, 31, 2)␣
 ↪#cityblock(également appelé distance de Manhattan)
parametres_grid={"n_neighbors":valeurs_de_k, 'metric' :distances}

grid=RandomizedSearchCV(neigh, parametres_grid)
start = time.time()
grid.fit(x_train,y_train)
print("randomized search took {:.2f} minutes".format((time.time() - start)/
 ↪60))#calculer le temps écoulé (en minutes)
acc = grid.score(x_test, y_test)
print("randomized search accuracy: {:.2f}%".format(acc * 100))#calculer la␣
 ↪précison en utilisant les paramètres optimaux
print("randomized search best parameters: {}".format(grid.best_params_))#les␣
 ↪paramètres optimaux
```

```
randomized search took 17.45 minutes
randomized search accuracy: 58.05%
randomized search best parameters: {'n_neighbors': 15, 'metric': 'cityblock'}
```

En utilisant GridSearchCV

[20]:
```
from sklearn.model_selection import GridSearchCV

grid = GridSearchCV(neigh, parametres_grid)
start = time.time()
grid.fit(x_train, y_train)
print("grid search took {:.2f} minutes".format((time.time() - start)/
 ↪60))#calculer le temps écoulé (en minutes)
acc = grid.score(x_test, y_test)
print("grid search accuracy: {:.2f}%".format(acc * 100))#calculer la précison␣
 ↪en utilisant les paramètres optimaux
print("grid search best parameters: {}".format(grid.best_params_))#les␣
 ↪paramètres optimaux (cityblock est Manhattan distance)
```

```
grid search took 77.90 minutes
grid search accuracy: 57.45%
grid search best parameters: {'metric': 'cityblock', 'n_neighbors': 23}
```

On remarque que l'utilisation de GridSearchCV a pris plus qu'une heure pour donner une précision de 57.45%, par contre en utilisant RandomizedSearchCV, on a obtenu une précision de 58.05% en presque 17 min seulement.

1.1.4 Le modèle de classification basé sur Decision Tree

[26]:
```
from sklearn.tree import DecisionTreeClassifier

#En utilisant l'indice de gini
clf_gini = DecisionTreeClassifier(criterion = "gini")
clf_gini.fit(x_train,y_train)
y_pred_gini = clf_gini.score(x_test,y_test)
print ("Accuracy_gini : {:.2f}%".format(y_pred_gini*100))

#En utilisant l'entropie
clf_entropy = DecisionTreeClassifier(criterion = "entropy")
clf_entropy.fit(x_train,y_train)
y_pred_entropy = clf_entropy.score(x_test,y_test)
print ("Accuracy_entropy : {:.2f}%".format(y_pred_entropy*100))
```

```
Accuracy_gini : 53.65%
Accuracy_entropy : 55.00%
```

Selon les resutats obtenu ,L'utilisation du KNN avec les paramètres optimaux a donné 58.05% alors que l'utilisation du DecisionTree(criterion = "entropy") a donné 55%.

On peut dire que le KNN performe mieux que le DecisionTree.

Note : En général, les algorithmes de machine learning ne sont pas bien adaptés à ce type de problème (traitement et classification des images) par contre, les algorithmes du deep learning (CNN par exemple) sont plus efficace et ils donnent des meilleures performances.