Котолевский Максим Николаевич, группа 19

Лабораторная работа № 4

**Вариант № 8**

## Цель работы

Классификация исходного изображения с помощью метода k ближайших соседей.

## Задание

Решить задачу классификации исходного изображения с помощью метода k ближайших соседей (цветы). Оценить точность полученной модели. Возможно использование преобученной нейронной сети.

**Код программы (внесённые изменения в шаблон кода выделены)**

Ссылка на исходный dataset:
https://www.kaggle.com/datasets/alxmamaev/flowers-recognition

```
//Загрузка необходимых библиотек и данных для обучения
import os
import numpy as np
import matplotlib.pyplot as plt

from sklearn import metrics
from sklearn.model_selection import train_test_split
from sklearn.neighbors import KNeighborsClassifier
from sklearn import random_projection
from tensorflow.keras.utils import to_categorical
from tensorflow.keras.preprocessing.image import load_img, img_to_array
from google.colab import drive

%matplotlib inline
drive.mount('/content/gdrive', force_remount=True)
RANDOM_STATE = 42

train_dir = "/content/gdrive/MyDrive/Colab Notebooks/data/train"
test_dir = "/content/gdrive/MyDrive/Colab Notebooks/data/test"
valid_dir = "/content/gdrive/MyDrive/Colab Notebooks/data/valid"

train_set = []
count = 0
for package in os.listdir(train_dir):
    path = os.path.join(train_dir, package)
    for im in os.listdir(path):
        image = load_img(os.path.join(path, im), target_size = (224,224,3))
        image = img_to_array(image)
        image = image / 255.0
        train_set += [[image,count]]
    count += 1

test_set = []
count = 0
```

```
for package in os.listdir(test_dir):
    path = os.path.join(test_dir, package)
    for im in os.listdir(path):
        image = load_img(os.path.join(path, im), target_size = (224,224,3))
        image = img_to_array(image)
        image = image / 255.0
        test_set += [[image,count]]
    count += 1

valid_set = []
count = 0
for package in os.listdir(valid_dir):
    path = os.path.join(valid_dir, package)
    for im in os.listdir(path):
        image = load_img(os.path.join(path, im), target_size = (224,224,3))
        image = img_to_array(image)
        image = image / 255.0
        valid_set += [[image,count]]
    count += 1

X_train, y_train = zip(*train_set)
X_test, y_test = zip(*test_set)
X_valid, y_valid = zip(*valid_set)

X_train = np.array(X_train)
X_test = np.array(X_test)
X_valid = np.array(X_valid)

y_train = np.array(to_categorical(y_train))
y_test = np.array(to_categorical(y_test))
y_valid = np.array(to_categorical(y_valid))

//Проверка данных для обучения
X_train.shape[0]

plt.figure(figsize = (10, 10))
i_subplot = 1
for i in range(X_train.shape[0]):
    plt.subplot(35, 35, i_subplot)
    i_subplot += 1
    plt.xticks([])
    plt.yticks([])
    plt.imshow(np.reshape(X_train[i, :], (224,224,3)))

train_class_names = []
for file in os.listdir(train_dir):
    train_class_names += [file]

labels = []
for item in y_train:
  labels.append(train_class_names[item.argmax()])
```

```
y_train = np.array(labels)
y_train

//Проверка данных для тестирования
plt.figure(figsize = (10, 10))
i_subplot = 1
for i in range(X_test.shape[0]):
    plt.subplot(25,25, i_subplot)
    i_subplot += 1
    plt.xticks([])
    plt.yticks([])
    plt.imshow(np.reshape(X_test[i, :], (224,224,3)))

test_class_names = []
for file in os.listdir(test_dir):
    test_class_names += [file]

labels = []
for item in y_test:
  labels.append(test_class_names[item.argmax()])

y_test = np.array(labels)
y_test

//Подготовка обучаемой, тестируемой и валидационной выборок
X_train = X_train.reshape((len(X_train), 224 * 224 * 3))
X_test = X_test.reshape((len(X_test), 224 * 224 * 3))
X_valid = X_valid.reshape((len(X_valid), 224 * 224 * 3))

val_class_names = []
for file in os.listdir(valid_dir):
    val_class_names += [file]

labels = []
for item in y_valid:
  labels.append(val_class_names[item.argmax()])

y_valid = np.array(labels)

//Выбор оптимального значения k метода KNN
model = KNeighborsClassifier(n_neighbors = 4)
model.fit(X_train, y_train)

y_test_pred = model.predict(X_test)

print('Accuracy on test data (n_neighbors = 4): ', metrics.accuracy_score(y_test, y_te
st_pred))
print('Loss on test data: (n_neighbors = 4):    ', np.mean(y_test != y_test_pred))

metrics.confusion_matrix(y_test, y_test_pred)
```

```
dict(zip(test_class_names, list(range(len(test_class_names)))))

M = metrics.confusion_matrix(y_test, y_test_pred)
M = np.sqrt(M)
plt.imshow(M, interpolation='nearest')
plt.xticks(range(len(test_class_names)))
plt.yticks(range(len(test_class_names)))
plt.xlabel("predicted label")
plt.ylabel("true label")
plt.colorbar()

kk = range(1, 15, 1)
err_train = []
err_test = []
for k in kk:
    model = KNeighborsClassifier(n_neighbors = k)
    model.fit(X_train, y_train)
    err_train.append(np.mean(model.predict(X_train) != y_train))
    err_test.append(np.mean(model.predict(X_test) != y_test))

plt.plot(kk, err_train, '.-r', label = 'Train error')
plt.plot(kk, err_test,  '.-b', label = 'Test error')
plt.xlabel('k')
plt.ylabel('error')
plt.legend(loc = 4)

min(err_test)

kk[err_test.index(min(err_test))]

//Обучение модели на оптимальном значении k и проверка на тестовой выборке
model = KNeighborsClassifier(n_neighbors = 7)
model.fit(X_train, y_train)
i=1
plt.figure(figsize = (20, 20))
i_subplot = 1
yi_test_pred = model.predict(X_test)
for i in range(X_test.shape[0]):
    if yi_test_pred[i] != y_test[i]:
        plt.subplot(10, 10, i_subplot)
        i_subplot += 1
        plt.xticks([])
        plt.yticks([])
        plt.imshow(np.reshape(X_test[i, :], (224,224,3)), cmap = plt.cm.binary)
        plt.text(0, 70, str(y_test[i]), color = 'b')
        plt.text(0, 10, str(yi_test_pred[i]), color = 'r')

print('Accuracy on test data (n_neighbors = 7): ', metrics.accuracy_score(y_test, yi_t
est_pred))
print('Loss on test data (n_neighbors = 7):     ', np.mean(y_test != yi_test_pred))
```

```
//Проверка модели на валидационной выборке
i=1
plt.figure(figsize = (20, 20))
i_subplot = 1
yi_valid_pred = model.predict(X_valid)
for i in range(X_valid.shape[0]):
    if yi_valid_pred[i] != y_valid[i]:
        plt.subplot(10, 10, i_subplot)
        i_subplot += 1
        plt.xticks([])
        plt.yticks([])
        plt.imshow(np.reshape(X_valid[i, :], (224,224,3)), cmap = plt.cm.binary)
        plt.text(0, 70, str(y_valid[i]), color = 'b')
        plt.text(0, 10, str(yi_valid_pred[i]), color = 'r')

print('Accuracy on validation data (n_neighbors = 7): ', metrics.accuracy_score(y_vali
d, yi_valid_pred))
print('Loss on validation data (n_neighbors = 7):     ', np.mean(y_valid != yi_valid_p
red))
```

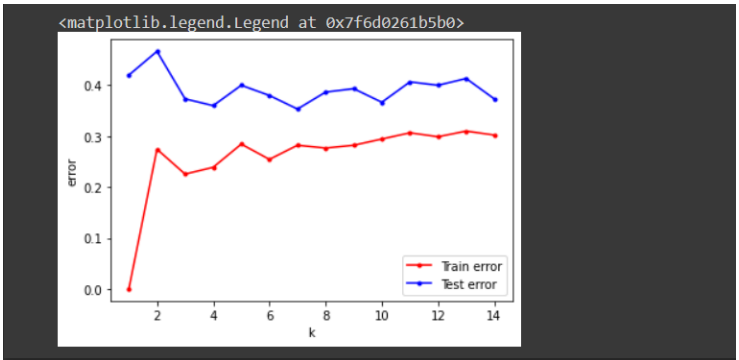
## Результаты выполнения задания

Сначала проводим обучение на рандомном количестве k (пример, на 4 соседях). Получаем:

```
[19] print('Accuracy on test data (n_neighbors = 4): ', metrics.accuracy_score(y_test, y_test_pred))
     print('Loss on test data: (n_neighbors = 4):    ', np.mean(y_test != y_test_pred))

     Accuracy on test data (n_neighbors = 4):  0.64
     Loss on test data: (n_neighbors = 4):     0.36
```

Улучшаем обучением нейронной сети на количестве соседей от 1 до 15 с шагом 1. Строим график и смотрим, какое значение минимальной ошибки получили и при каком количестве соседей.




```
[25] min(err_test)

     0.35333333333333333

[26] kk[err_test.index(min(err_test))]

     7
```

Теперь обучаем модель при оптимальном k.

Проверяем значения на тестовой выборке:

```
[40] print('Accuracy on test data (n_neighbors = 7): ', metrics.accuracy_score(y_test, yi_test_pred))
     print('Loss on test data (n_neighbors = 7):     ', np.mean(y_test != yi_test_pred))

     Accuracy on test data (n_neighbors = 7):  0.6466666666666666
     Loss on test data (n_neighbors = 7):      0.35333333333333333
```

Проверяем значения на валидационной выборке:

```
print('Accuracy on validation data (n_neighbors = 7): ', metrics.accuracy_score(y_valid, yi_valid_pred))
print('Loss on validation data (n_neighbors = 7):     ', np.mean(y_valid != yi_valid_pred))

Accuracy on validation data (n_neighbors = 7):  0.58
Loss on validation data (n_neighbors = 7):      0.42
```