Лабораторная работа №5

Применение сверточных нейронных сетей (бинарная классификация)

Набор данных *DogsVsCats*, который состоит из изображений различной размерности, содержащих фотографии собак и кошек.

Обучающая выборка включает в себя 25 тыс. изображений (12,5 тыс. кошек: *cat.0.jpg*, ..., *cat.12499.jpg* и 12,5 тыс. собак: *dog.0.jpg*, …, *dog.12499.jpg*), а контрольная выборка содержит 12,5 тыс. неразмеченных изображений.

Скачать данные, а также проверить качество классификатора на тестовой выборке можно на сайте *Kaggle*: https://www.kaggle.com/c/dogs-vs-cats/data (https://www.kaggle.com/c/dogs-vs-cats/data)

Задание 1

Загрузите данные. Разделите исходный набор данных на обучающую, валидационную и контрольную выборки.

In [1]:

```
from google.colab import drive

drive.mount('/content/drive', force_remount = True)
```

```
Mounted at /content/drive
```

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2/dogs-vs-cats'

import sys

sys.path.append(BASE_DIR)

import os
```

In [0]:

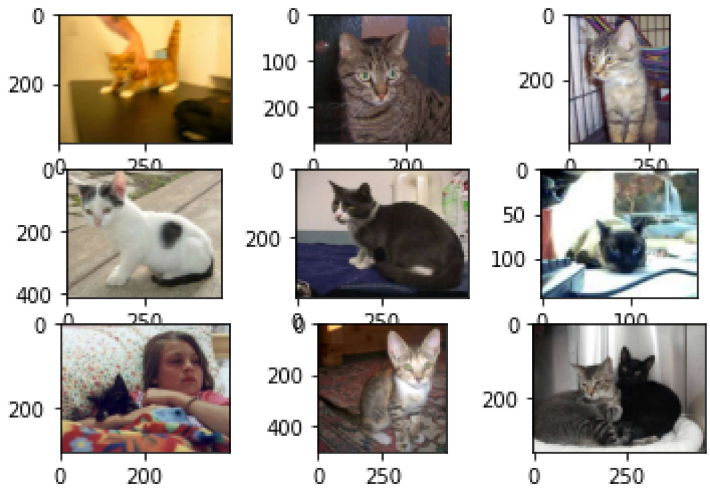
```
TRAIN_ARCHIVE_NAME = 'train.zip'
TEST_ARCHIVE_NAME = 'test1.zip'

LOCAL_DIR_NAME = 'dogs-vs-cats'
```

In [0]:

```
from zipfile import ZipFile

with ZipFile(os.path.join(BASE_DIR, TRAIN_ARCHIVE_NAME), 'r') as zip_:
    zip_.extractall(path = os.path.join(LOCAL_DIR_NAME, 'train'))

with ZipFile(os.path.join(BASE_DIR, TEST_ARCHIVE_NAME), 'r') as zip_:
    zip_.extractall(path = os.path.join(LOCAL_DIR_NAME, 'test-1'))
```

In [5]:

```
from matplotlib import pyplot
from matplotlib.image import imread

dir_ = 'dogs-vs-cats/train/train'

for i in range(9):

    pyplot.subplot(330 + 1 + i)

    image_ = imread('{}/cat.{}.jpg'.format(dir_, i))

    pyplot.imshow(image_)

pyplot.show()
```



In [0]:

```
NEW_IMAGE_WIDTH = 100
```

In [7]:

```
from os import listdir
from os.path import join
from numpy import asarray
from numpy import save
from keras.preprocessing.image import load_img
from keras.preprocessing.image import img_to_array

def dir_to_dataset(_dir_path):

    photos_, labels_ = [], []

    for file_ in listdir(_dir_path):

        if file_.startswith('cat'):
            label_ = 1.0
        else:
            label_ = 0.0

        photo_ = load_img(join(_dir_path, file_), target_size = (NEW_IMAGE_WIDTH, NEW_IMAGE

        photo_ = img_to_array(photo_)

        photos_.append(photo_)
        labels_.append(label_)

    photos_norm_ = tf.keras.utils.normalize(photos_, axis = 1)

    return asarray(photos_norm_), asarray(labels_)
```

```
Using TensorFlow backend.
```

In [8]:

```
! pip install tensorflow-gpu --pre --quiet

! pip show tensorflow-gpu
```

```
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: grpcio, h5py, gast, numpy, protobuf, google-pasta, absl-py, astunp
arse, opt-einsum, six, wheel, scipy, tensorflow-estimator, keras-preprocessi
ng, tensorboard, termcolor, wrapt
Required-by:
```

In [0]:

```
import tensorflow as tf
```

In [0]:

```
import numpy as np
```

In [0]:

```
X_all, y_all = dir_to_dataset('dogs-vs-cats/train/train')
```

In [0]:

```
TEST_LEN_HALF = 1000
```

In [13]:

```
test_interval = np.r_[0:TEST_LEN_HALF, -TEST_LEN_HALF:-0]

X, y = X_all[TEST_LEN_HALF:-TEST_LEN_HALF], y_all[TEST_LEN_HALF:-TEST_LEN_HALF]
X_test, y_test = X_all[test_interval], y_all[test_interval]

print(X.shape, y.shape)
print(X_test.shape, y_test.shape)
```

```
(23000, 100, 100, 3) (23000,)
(2000, 100, 100, 3) (2000,)
```

Выделение валидационной выборки произойдёт автоматически по параметру `validation_split` метода `model.fit()` .

Задание 2

Реализуйте глубокую нейронную сеть с как минимум тремя сверточными слоями. Какое качество классификации получено?

In [0]:

```
from tensorflow import keras
```

In [15]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, MaxPooling2D, Flatten, Dense

model = tf.keras.Sequential()

model.add(Conv2D(16, 3, padding = 'same', activation = 'relu', input_shape = (NEW_IMAGE_WID
model.add(MaxPooling2D())
model.add(Conv2D(32, 3, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D())
model.add(Conv2D(64, 3, padding = 'same', activation = 'relu'))
model.add(MaxPooling2D())
model.add(Flatten())
model.add(Dense(512, activation = 'relu'))
model.add(Dense(1, activation = 'sigmoid'))

model.compile(optimizer = 'sgd',
              loss = 'binary_crossentropy',
              metrics = ['accuracy'])

model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d (Conv2D)              (None, 100, 100, 16)      448
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 50, 50, 16)        0
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 50, 50, 32)        4640
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4719104
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,743,201
Trainable params: 4,743,201
Non-trainable params: 0
_________________________________________________________________
```

In [16]:

```
model.fit(x = X, y = y, epochs = 20, validation_split = 0.15)
```

```
Epoch 1/20
611/611 [==============================] - 5s 8ms/step - loss: 0.6907 - accu
racy: 0.5365 - val_loss: 0.6885 - val_accuracy: 0.5670
Epoch 2/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6858 - accu
racy: 0.5711 - val_loss: 0.6845 - val_accuracy: 0.5481
Epoch 3/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6794 - accu
racy: 0.5812 - val_loss: 0.6754 - val_accuracy: 0.5957
Epoch 4/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6724 - accu
racy: 0.5894 - val_loss: 0.6685 - val_accuracy: 0.5875
Epoch 5/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6626 - accu
racy: 0.6043 - val_loss: 0.6565 - val_accuracy: 0.6104
Epoch 6/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6529 - accu
racy: 0.6184 - val_loss: 0.6447 - val_accuracy: 0.6301
Epoch 7/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6425 - accu
racy: 0.6298 - val_loss: 0.6570 - val_accuracy: 0.6038
Epoch 8/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6313 - accu
racy: 0.6492 - val_loss: 0.6229 - val_accuracy: 0.6600
Epoch 9/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6184 - accu
racy: 0.6617 - val_loss: 0.6195 - val_accuracy: 0.6501
Epoch 10/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6086 - accu
racy: 0.6730 - val_loss: 0.6173 - val_accuracy: 0.6562
Epoch 11/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5992 - accu
racy: 0.6808 - val_loss: 0.6338 - val_accuracy: 0.6336
Epoch 12/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5900 - accu
racy: 0.6877 - val_loss: 0.6000 - val_accuracy: 0.6771
Epoch 13/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5791 - accu
racy: 0.6984 - val_loss: 0.5896 - val_accuracy: 0.6829
Epoch 14/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5655 - accu
racy: 0.7121 - val_loss: 0.5739 - val_accuracy: 0.6980
Epoch 15/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5529 - accu
racy: 0.7184 - val_loss: 0.5741 - val_accuracy: 0.7041
Epoch 16/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5407 - accu
racy: 0.7278 - val_loss: 0.5559 - val_accuracy: 0.7113
Epoch 17/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5239 - accu
racy: 0.7384 - val_loss: 0.5465 - val_accuracy: 0.7235
Epoch 18/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5113 - accu
racy: 0.7481 - val_loss: 0.5411 - val_accuracy: 0.7365
Epoch 19/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4984 - accu
racy: 0.7573 - val_loss: 0.5382 - val_accuracy: 0.7374
```

```
Epoch 20/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4847 - accu
racy: 0.7671 - val_loss: 0.5281 - val_accuracy: 0.7403
```

Out[16]:

```
<tensorflow.python.keras.callbacks.History at 0x7f31f0e59198>
```

In [17]:

```
results = model.evaluate(X_test, y_test)

print('Test loss, test accuracy:', results)
```

```
63/63 [==============================] - 0s 4ms/step - loss: 0.5323 - accura
cy: 0.7260
Test loss, test accuracy: [0.5323428511619568, 0.7260000109672546]
```

Результат — 72% на тестовой выборке.

Задание 3

Примените дополнение данных (*data augmentation*). Как это повлияло на качество классификатора?

Задание 4

Поэкспериментируйте с готовыми нейронными сетями (например, *AlexNet*, *VGG16*, *Inception* и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора?

Какой максимальный результат удалось получить на сайте *Kaggle*? Почему?