Лабораторная работа №4

Реализация приложения по распознаванию номеров домов

Набор изображений из *Google Street View* с изображениями номеров домов, содержащий 10 классов, соответствующих цифрам от 0 до 9.

- 73257 изображений цифр в обучающей выборке;
- 26032 изображения цифр в тестовой выборке;
- 531131 изображения, которые можно использовать как дополнение к обучающей выборке;
- В двух форматах:
  - Оригинальные изображения с выделенными цифрами;
  - Изображения размером 32×32, содержащие одну цифру;
- Данные первого формата можно скачать по ссылкам:
  - http://ufldl.stanford.edu/housenumbers/train.tar.gz (http://ufldl.stanford.edu/housenumbers/train.tar.gz) (обучающая выборка);
  - http://ufldl.stanford.edu/housenumbers/test.tar.gz (http://ufldl.stanford.edu/housenumbers/test.tar.gz) (тестовая выборка);
  - http://ufldl.stanford.edu/housenumbers/extra.tar.gz (http://ufldl.stanford.edu/housenumbers/extra.tar.gz) (дополнительные данные);
- Данные второго формата можно скачать по ссылкам:
  - http://ufldl.stanford.edu/housenumbers/train_32x32.mat (http://ufldl.stanford.edu/housenumbers/train_32x32.mat) (обучающая выборка);
  - http://ufldl.stanford.edu/housenumbers/test_32x32.mat (http://ufldl.stanford.edu/housenumbers/test_32x32.mat) (тестовая выборка);
  - http://ufldl.stanford.edu/housenumbers/extra_32x32.mat (http://ufldl.stanford.edu/housenumbers/extra_32x32.mat) (дополнительные данные);
- Описание данных на английском языке доступно по ссылке:
  - http://ufldl.stanford.edu/housenumbers/ (http://ufldl.stanford.edu/housenumbers/)

Задание 1

Реализуйте глубокую нейронную сеть (полносвязную или сверточную) и обучите ее на синтетических данных (например, наборы *MNIST* (http://yann.lecun.com/exdb/mnist/ (http://yann.lecun.com/exdb/mnist/)) или *notMNIST*).

Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf)), видео на *YouTube* (https://www.youtube.com/watch?v=vGPI_JvLoN0) (https://www.youtube.com/watch?v=vGPI_JvLoN0)).

Используем архитектуру *LeNet-5* и обучим сеть сначала на данных из набора *MNIST*.

In [1]:

```
! 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, tensorboard, termcolor, six, gast, absl-py, tensorflow-est
imator, opt-einsum, wrapt, wheel, numpy, google-pasta, protobuf, scipy, astu
nparse, keras-preprocessing, h5py
Required-by:
```

In [0]:

```
import tensorflow as tf
from tensorflow import keras
```

In [0]:

```
import numpy as np
```

In [0]:

```
from tensorflow.keras.datasets import mnist

(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [0]:

```
x_train, x_test = tf.keras.utils.normalize(x_train, axis = 1), tf.keras.utils.normalize(x_t
```

In [0]:

```
x_train, x_test = x_train[..., np.newaxis], x_test[..., np.newaxis]
```

In [7]:

```
from tensorflow.keras.utils import to_categorical

y_train, y_test = to_categorical(y_train), to_categorical(y_test)

y_train.shape
```

Out[7]:

```
(60000, 10)
```

In [0]:

```
IMAGE_DIM_0, IMAGE_DIM_1 = x_train.shape[1], x_train.shape[2]
```

In [0]:

```
CLASSES_N = y_train.shape[1]
```

In [10]:

```
x_train.shape, x_test.shape
```

Out[10]:

```
((60000, 28, 28, 1), (10000, 28, 28, 1))
```

In [0]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import AveragePooling2D, Conv2D, Dense, Flatten

model = tf.keras.Sequential()

model.add(Conv2D(6, kernel_size = (5, 5), strides = (1, 1), activation = 'tanh', padding =
                   input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)))
model.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
model.add(Conv2D(16, kernel_size = (5, 5), strides = (1, 1), activation = 'tanh', padding =
model.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
model.add(Flatten())
model.add(Dense(120, activation = 'tanh'))
model.add(Dense(84, activation = 'tanh'))
model.add(Dense(CLASSES_N, activation = 'softmax'))
```

In [0]:

```
# 'sparse_categorical_crossentropy' gave NAN loss

model.compile(optimizer = 'adam',
              loss = 'categorical_crossentropy',
              metrics = ['categorical_accuracy'])
```

In [13]:

```
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 28, 28, 6)         156       
_________________________________________________________________
average_pooling2d (AveragePo (None, 14, 14, 6)         0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 10, 10, 16)        2416      
_________________________________________________________________
average_pooling2d_1 (Average (None, 5, 5, 16)          0         
_________________________________________________________________
flatten (Flatten)            (None, 400)               0         
_________________________________________________________________
dense (Dense)                (None, 120)               48120     
_________________________________________________________________
dense_1 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_2 (Dense)              (None, 10)                850       
=================================================================
Total params: 61,706
Trainable params: 61,706
Non-trainable params: 0
_________________________________________________________________
```

In [0]:

```
EPOCHS_N = 20
```

In [15]:

```
model.fit(x = x_train, y = y_train, validation_split = 0.15, epochs = EPOCHS_N)
```

```
Epoch 1/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.2895 - ca
tegorical_accuracy: 0.9129 - val_loss: 0.1330 - val_categorical_accuracy: 0.
9591
Epoch 2/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.1186 - ca
tegorical_accuracy: 0.9639 - val_loss: 0.0995 - val_categorical_accuracy: 0.
9694
Epoch 3/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0801 - ca
tegorical_accuracy: 0.9748 - val_loss: 0.0960 - val_categorical_accuracy: 0.
9697
Epoch 4/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0631 - ca
tegorical_accuracy: 0.9799 - val_loss: 0.0737 - val_categorical_accuracy: 0.
9786
Epoch 5/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0484 - ca
tegorical_accuracy: 0.9848 - val_loss: 0.0679 - val_categorical_accuracy: 0.
9802
Epoch 6/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0394 - ca
tegorical_accuracy: 0.9874 - val_loss: 0.0647 - val_categorical_accuracy: 0.
9806
Epoch 7/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0328 - ca
tegorical_accuracy: 0.9894 - val_loss: 0.0688 - val_categorical_accuracy: 0.
9811
Epoch 8/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0268 - ca
tegorical_accuracy: 0.9919 - val_loss: 0.0698 - val_categorical_accuracy: 0.
9806
Epoch 9/20
1594/1594 [==============================] - 4s 3ms/step - loss: 0.0242 - ca
tegorical_accuracy: 0.9921 - val_loss: 0.0617 - val_categorical_accuracy: 0.
9822
Epoch 10/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0197 - ca
tegorical_accuracy: 0.9935 - val_loss: 0.0687 - val_categorical_accuracy: 0.
9822
Epoch 11/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0168 - ca
tegorical_accuracy: 0.9944 - val_loss: 0.0719 - val_categorical_accuracy: 0.
9811
Epoch 12/20
1594/1594 [==============================] - 4s 3ms/step - loss: 0.0155 - ca
tegorical_accuracy: 0.9948 - val_loss: 0.0680 - val_categorical_accuracy: 0.
9808
Epoch 13/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0137 - ca
tegorical_accuracy: 0.9955 - val_loss: 0.0679 - val_categorical_accuracy: 0.
9822
Epoch 14/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0121 - ca
tegorical_accuracy: 0.9965 - val_loss: 0.0768 - val_categorical_accuracy: 0.
9806
Epoch 15/20
```

```
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0115 - ca
tegorical_accuracy: 0.9961 - val_loss: 0.0747 - val_categorical_accuracy: 0.
9824
Epoch 16/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0122 - ca
tegorical_accuracy: 0.9958 - val_loss: 0.0687 - val_categorical_accuracy: 0.
9840
Epoch 17/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0095 - ca
tegorical_accuracy: 0.9966 - val_loss: 0.0633 - val_categorical_accuracy: 0.
9848
Epoch 18/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0100 - ca
tegorical_accuracy: 0.9965 - val_loss: 0.0701 - val_categorical_accuracy: 0.
9836
Epoch 19/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0055 - ca
tegorical_accuracy: 0.9984 - val_loss: 0.0706 - val_categorical_accuracy: 0.
9842
Epoch 20/20
1594/1594 [==============================] - 5s 3ms/step - loss: 0.0093 - ca
tegorical_accuracy: 0.9967 - val_loss: 0.0714 - val_categorical_accuracy: 0.
9838
```

Out[15]:

```
<tensorflow.python.keras.callbacks.History at 0x7f60025300f0>
```

In [16]:

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

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

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0828 - cate
gorical_accuracy: 0.9808
Test loss, test accuracy: [0.08280119299888611, 0.9807999730110168]
```

Удалось достичь отличного результата — точность распознавания на тестовой выборке составила 98,0%.

Задание 2

После уточнения модели на синтетических данных попробуйте обучить ее на реальных данных (набор *Google Street View*). Что изменилось в модели?

In [0]:

```
DS_URL_FOLDER = 'http://ufldl.stanford.edu/housenumbers/'

FIRST_DS_EXT = '.tar.gz'
SECOND_DS_EXT = '_32x32.mat'

TRAIN_DS_NAME = 'train'
TEST_DS_NAME = 'test'
EXTRA_DS_NAME = 'extra'
```

In [0]:

```
from urllib.request import urlretrieve
import tarfile
import os

def load_file(_url_folder, _name, _ext, _key, _local_ext = ''):

    file_url_ = _url_folder + _name + _ext

    local_file_name_ = _name + '_' + _key + _local_ext

    urlretrieve(file_url_, local_file_name_)

    return local_file_name_

def tar_gz_to_dir(_url_folder, _name, _ext, _key):

    local_file_name_ = load_file(_url_folder, _name, _ext, _key, _ext)

    dir_name_ = _name + '_' + _key

    with tarfile.open(local_file_name_, 'r:gz') as tar_:
        tar_.extractall(dir_name_)

    os.remove(local_file_name_)

    return dir_name_
```

In [0]:

```
first_ds_train_dir = tar_gz_to_dir(DS_URL_FOLDER, TRAIN_DS_NAME, FIRST_DS_EXT, 'first')
first_ds_test_dir = tar_gz_to_dir(DS_URL_FOLDER, TEST_DS_NAME, FIRST_DS_EXT, 'first')
first_ds_extra_dir = tar_gz_to_dir(DS_URL_FOLDER, EXTRA_DS_NAME, FIRST_DS_EXT, 'first')
```

In [0]:

```
second_ds_train_file = load_file(DS_URL_FOLDER, TRAIN_DS_NAME, SECOND_DS_EXT, 'second')
second_ds_test_file = load_file(DS_URL_FOLDER, TEST_DS_NAME, SECOND_DS_EXT, 'second')
second_ds_extra_file = load_file(DS_URL_FOLDER, EXTRA_DS_NAME, SECOND_DS_EXT, 'second')
```

In [0]:

```
from scipy import io

second_ds_train = io.loadmat(second_ds_train_file)
second_ds_test = io.loadmat(second_ds_test_file)
second_ds_extra = io.loadmat(second_ds_extra_file)
```

In [22]:

```
X_second_ds_train = np.moveaxis(second_ds_train['X'], -1, 0)
X_second_ds_test = np.moveaxis(second_ds_test['X'], -1, 0)
X_second_ds_extra = np.moveaxis(second_ds_extra['X'], -1, 0)

y_second_ds_train = second_ds_train['y']
y_second_ds_test = second_ds_test['y']
y_second_ds_extra = second_ds_extra['y']

print(X_second_ds_train.shape, y_second_ds_train.shape)
print(X_second_ds_test.shape, y_second_ds_test.shape)
print(X_second_ds_extra.shape, y_second_ds_extra.shape)
```

```
(73257, 32, 32, 3) (73257, 1)
(26032, 32, 32, 3) (26032, 1)
(531131, 32, 32, 3) (531131, 1)
```

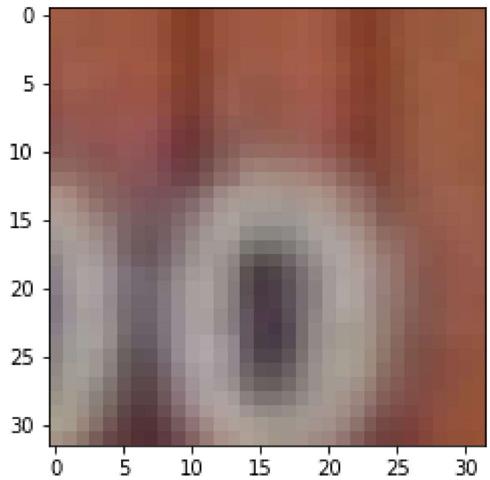
In [23]:

```
import matplotlib.pyplot as plt

plt.imshow(X_second_ds_train[100])
plt.imshow(X_second_ds_test[100])
plt.imshow(X_second_ds_extra[100])
```

Out[23]:

```
&lt;matplotlib.image.AxesImage at 0x7f6002399320&gt;
```



In [0]:

```
IMAGE_DIM_0_2, IMAGE_DIM_1_2, IMAGE_DIM_2_2 = X_second_ds_train.shape[-3], X_second_ds_trai
```

In [0]:

```
y_second_ds_train_cat = to_categorical(y_second_ds_train)
y_second_ds_test_cat = to_categorical(y_second_ds_test)
```

In [0]:

```
CLASSES_N_2 = y_second_ds_train_cat.shape[1]
```

In [0]:

```
model_2 = tf.keras.Sequential()

model_2.add(Conv2D(6, kernel_size = (5, 5), strides = (1, 1), activation = 'tanh', padding
                   input_shape = (IMAGE_DIM_0_2, IMAGE_DIM_1_2, IMAGE_DIM_2_2)))
model_2.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
model_2.add(Conv2D(16, kernel_size = (5, 5), strides = (1, 1), activation = 'tanh', padding
model_2.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2), padding = 'valid'))
model_2.add(Flatten())
model_2.add(Dense(120, activation = 'tanh'))
model_2.add(Dense(84, activation = 'tanh'))
model_2.add(Dense(CLASSES_N_2, activation = 'softmax'))
```

In [0]:

```
model_2.compile(optimizer = 'adam',
                loss = 'categorical_crossentropy',
                metrics = ['categorical_accuracy'])
```

In [29]:

```
model_2.summary()
```

```
Model: "sequential_1"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d_2 (Conv2D)            (None, 32, 32, 6)         456       
_________________________________________________________________
average_pooling2d_2 (Average (None, 16, 16, 6)         0         
_________________________________________________________________
conv2d_3 (Conv2D)            (None, 12, 12, 16)        2416      
_________________________________________________________________
average_pooling2d_3 (Average (None, 6, 6, 16)          0         
_________________________________________________________________
flatten_1 (Flatten)          (None, 576)               0         
_________________________________________________________________
dense_3 (Dense)              (None, 120)               69240     
_________________________________________________________________
dense_4 (Dense)              (None, 84)                10164     
_________________________________________________________________
dense_5 (Dense)              (None, 11)                935       
=================================================================
Total params: 83,211
Trainable params: 83,211
Non-trainable params: 0
_________________________________________________________________
```

In [30]:

```
model_2.fit(x = X_second_ds_train, y = y_second_ds_train_cat, validation_split = 0.15, epoc
```

```
Epoch 1/20
1946/1946 [==============================] - 6s 3ms/step - loss: 1.1878 - ca
tegorical_accuracy: 0.6141 - val_loss: 0.8286 - val_categorical_accuracy: 0.
7351
Epoch 2/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.6917 - ca
tegorical_accuracy: 0.7843 - val_loss: 0.6128 - val_categorical_accuracy: 0.
8132
Epoch 3/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.5979 - ca
tegorical_accuracy: 0.8130 - val_loss: 0.5611 - val_categorical_accuracy: 0.
8280
Epoch 4/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.5475 - ca
tegorical_accuracy: 0.8287 - val_loss: 0.5569 - val_categorical_accuracy: 0.
8276
Epoch 5/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.5193 - ca
tegorical_accuracy: 0.8383 - val_loss: 0.5476 - val_categorical_accuracy: 0.
8333
Epoch 6/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.4886 - ca
tegorical_accuracy: 0.8474 - val_loss: 0.5300 - val_categorical_accuracy: 0.
8375
Epoch 7/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.4755 - ca
tegorical_accuracy: 0.8511 - val_loss: 0.5751 - val_categorical_accuracy: 0.
8284
Epoch 8/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.4482 - ca
tegorical_accuracy: 0.8613 - val_loss: 0.5043 - val_categorical_accuracy: 0.
8469
Epoch 9/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.4372 - ca
tegorical_accuracy: 0.8642 - val_loss: 0.5296 - val_categorical_accuracy: 0.
8393
Epoch 10/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.4162 - ca
tegorical_accuracy: 0.8697 - val_loss: 0.5304 - val_categorical_accuracy: 0.
8381
Epoch 11/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.4058 - ca
tegorical_accuracy: 0.8733 - val_loss: 0.5445 - val_categorical_accuracy: 0.
8385
Epoch 12/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.4027 - ca
tegorical_accuracy: 0.8753 - val_loss: 0.5371 - val_categorical_accuracy: 0.
8347
Epoch 13/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.3966 - ca
tegorical_accuracy: 0.8747 - val_loss: 0.5472 - val_categorical_accuracy: 0.
8381
Epoch 14/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.3922 - ca
tegorical_accuracy: 0.8759 - val_loss: 0.5105 - val_categorical_accuracy: 0.
8482
Epoch 15/20
```

```
1946/1946 [==============================] - 5s 3ms/step - loss: 0.3835 - ca
tegorical_accuracy: 0.8784 - val_loss: 0.5198 - val_categorical_accuracy: 0.
8477
Epoch 16/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.3669 - ca
tegorical_accuracy: 0.8850 - val_loss: 0.4903 - val_categorical_accuracy: 0.
8523
Epoch 17/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.3616 - ca
tegorical_accuracy: 0.8858 - val_loss: 0.5543 - val_categorical_accuracy: 0.
8359
Epoch 18/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.3559 - ca
tegorical_accuracy: 0.8873 - val_loss: 0.5141 - val_categorical_accuracy: 0.
8467
Epoch 19/20
1946/1946 [==============================] - 6s 3ms/step - loss: 0.3472 - ca
tegorical_accuracy: 0.8914 - val_loss: 0.5160 - val_categorical_accuracy: 0.
8511
Epoch 20/20
1946/1946 [==============================] - 5s 3ms/step - loss: 0.3439 - ca
tegorical_accuracy: 0.8917 - val_loss: 0.6758 - val_categorical_accuracy: 0.
7925
```

Out[30]:

```
<tensorflow.python.keras.callbacks.History at 0x7f60022ca4e0>
```

In [31]:

```
results = model_2.evaluate(X_second_ds_test, y_second_ds_test_cat)

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

```
814/814 [==============================] - 1s 2ms/step - loss: 0.7207 - cate
gorical_accuracy: 0.7746
Test loss, test accuracy: [0.7207155227661133, 0.7745851278305054]
```

Прежде всего, в модели изменилось то, что добавился ещё один класс — *не распознано*.

Эти данные более сложны для распознавания, что повлияло на результат — точность распознавания на тестовой выборке составила 77,4%.

Задание 3

Сделайте множество снимков изображений номеров домов с помощью смартфона на ОС *Android*. Также можно использовать библиотеки *OpenCV*, *Simple CV* или *Pygame* для обработки изображений с общедоступных камер видеонаблюдения (например, [https://www.earthcam.com/) (https://www.earthcam.com/)](https://www.earthcam.com/)).

В качестве примера использования библиотеки *TensorFlow* на смартфоне можете воспользоваться демонстрационным приложением от *Google* ([https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android) (https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android)](https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/android)).

Задание 4

Реализуйте приложение для ОС *Android*, которое может распознавать цифры в номерах домов, используя разработанный ранее классификатор. Какова доля правильных классификаций?