Лабораторная работа №4. Реализация приложения по распознаванию номеров домов.

Данные: 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, содержащих одну цифру; Данные первого формата можно скачать по ссылкам: http://ufldl.stanford.edu/housenumbers/train.tar.gz (обучающая выборка); http://ufldl.stanford.edu/housenumbers/test.tar.gz (тестовая выборка); 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 (обучающая выборка); http://ufldl.stanford.edu/housenumbers/test_32x32.mat (тестовая выборка); http://ufldl.stanford.edu/housenumbers/extra_32x32.mat (дополнительные данные); Описание данных на английском языке доступно по ссылке: 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/) или notMNIST). Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf), видео на YouTube (https://www.youtube.com/watch?v=vGPI_JvLoN0).

In [1]:

```
from tensorflow import keras
import numpy as np
import scipy.io
import os
import h5py
from PIL import Image

max_len = 6
number_of_digits = 10
wildcard = 10

epochs = 5

data_folder = '../data'
```

In [2]:

```
(X_train, y_train), (X_dev, y_dev) = keras.datasets.mnist.load_data()

X_train = X_train.reshape((X_train.shape + (1, )))
X_dev = X_dev.reshape((X_dev.shape + (1, )))

y_train = y_train.reshape((y_train.shape + (1, )))
y_dev = y_dev.reshape((y_dev.shape + (1, )))
```

In [3]:

```
def preprocess(y):
    result = [[] for _ in range(max_len + 1)]
    for element in y:
        digit = str(element[0])

        result[0].append(len(digit))

        for j in range (max_len):
            val = int(digit[j]) if j < len(digit) else wildcard
            result[j + 1].append(val)

    return [np.array(output).reshape((len(y), 1)) for output in result]
```

In [4]:

```
y_train = preprocess(y_train)
```

```
y_dev = preprocess(y_dev)
```

In [5]:

```
def le_net_5_modern_model(input_shape):
    inputs = keras.layers.Input(shape=input_shape)

    hidden1 = keras.layers.Conv2D(6, (3, 3), activation='relu')(inputs)
    hidden1 = keras.layers.MaxPooling2D()(hidden1)

    hidden2 = keras.layers.Conv2D(16, (3, 3), activation='relu')(hidden1)
    hidden2 = keras.layers.MaxPooling2D(padding="same")(hidden2)

    hidden3 = keras.layers.Flatten()(hidden2)
    hidden3 = keras.layers.Dense(120, activation='relu')(hidden3)

    hidden4 = keras.layers.Dense(84, activation='relu')(hidden3)

    max_len_output = keras.layers.Dense(max_len + 1, activation = 'softmax')(hidden4)
    digit_outputs = [keras.layers.Dense(number_of_digits + 1, activation = 'softmax')(hidden4)
                     for _ in range(max_len)]

    return keras.Model(inputs=inputs, outputs=[max_len_output] + digit_outputs)
```

In [6]:

```
def train(model, epochs):
    model.compile(optimizer='adam',
                  loss='sparse_categorical_crossentropy',
                  metrics=['accuracy'])

    model.fit(X_train, y_train, epochs=epochs, batch_size=256)

    return model.evaluate(X_dev, y_dev)
```

In [7]:

```
train(le_net_5_modern_model(X_train.shape[1:]), epochs)
```

```
Train on 60000 samples
Epoch 1/5
60000/60000 [==============================] - 35s 584us/sample - loss: 5.3751 - dense_2_loss: 0.0
698 - dense_3_loss: 2.6253 - dense_4_loss: 0.3824 - dense_5_loss: 1.0715 - dense_6_loss: 0.2605 -
dense_7_loss: 0.1635 - dense_8_loss: 0.7897 - dense_2_accuracy: 0.9941 - dense_3_accuracy: 0.7749
- dense_4_accuracy: 0.9858 - dense_5_accuracy: 0.9822 - dense_6_accuracy: 0.9891 -
dense_7_accuracy: 0.9909 - dense_8_accuracy: 0.9844
Epoch 2/5
60000/60000 [==============================] - 34s 561us/sample - loss: 0.2633 - dense_2_loss: 0.0
000e+00 - dense_3_loss: 0.2630 - dense_4_loss: 9.7298e-07 - dense_5_loss: 8.6456e-05 -
dense_6_loss: 3.1257e-05 - dense_7_loss: 1.4265e-05 - dense_8_loss: 8.6011e-05 - dense_2_accuracy:
1.0000 - dense_3_accuracy: 0.9359 - dense_4_accuracy: 1.0000 - dense_5_accuracy: 1.0000 -
dense_6_accuracy: 1.0000 - dense_7_accuracy: 1.0000 - dense_8_accuracy: 1.0000
Epoch 3/5
60000/60000 [==============================] - 32s 528us/sample - loss: 0.1600 - dense_2_loss: 0.0
000e+00 - dense_3_loss: 0.1596 - dense_4_loss: 2.7721e-06 - dense_5_loss: 5.3686e-05 -
dense_6_loss: 2.5267e-05 - dense_7_loss: 3.3568e-05 - dense_8_loss: 3.1583e-05 - dense_2_accuracy:
1.0000 - dense_3_accuracy: 0.9570 - dense_4_accuracy: 1.0000 - dense_5_accuracy: 1.0000 -
dense_6_accuracy: 1.0000 - dense_7_accuracy: 1.0000 - dense_8_accuracy: 1.0000
Epoch 4/5
60000/60000 [==============================] - 32s 536us/sample - loss: 0.1154 - dense_2_loss: 3.9
631e-12 - dense_3_loss: 0.1152 - dense_4_loss: 1.9085e-05 - dense_5_loss: 3.2637e-05 -
dense_6_loss: 2.3285e-05 - dense_7_loss: 3.6202e-05 - dense_8_loss: 1.3392e-05 - dense_2_accuracy:
1.0000 - dense_3_accuracy: 0.9682 - dense_4_accuracy: 1.0000 - dense_5_accuracy: 1.0000 -
dense_6_accuracy: 1.0000 - dense_7_accuracy: 1.0000 - dense_8_accuracy: 1.0000
Epoch 5/5
60000/60000 [==============================] - 32s 538us/sample - loss: 0.0902 - dense_2_loss: 1.1
889e-11 - dense_3_loss: 0.0901 - dense_4_loss: 7.5113e-06 - dense_5_loss: 4.3943e-05 -
dense_6_loss: 2.2469e-05 - dense_7_loss: 5.2991e-05 - dense_8_loss: 7.6469e-06 - dense_2_accuracy:
1.0000 - dense_3_accuracy: 0.9740 - dense_4_accuracy: 1.0000 - dense_5_accuracy: 1.0000 -
dense_6_accuracy: 1.0000 - dense_7_accuracy: 1.0000 - dense_8_accuracy: 1.0000
10000/10000 [==============================] - 4s 356us/sample - loss: 0.1127 - dense_2_loss:
1.0712e-10 - dense_3_loss: 0.1124 - dense_4_loss: 3.5252e-06 - dense_5_loss: 4.2240e-05 -
dense_6_loss: 3.9821e-05 - dense_7_loss: 6.1822e-06 - dense_8_loss: 3.0734e-07 - dense_2_accuracy:
```

```
1.0000 - dense_3_accuracy: 0.9730 - dense_4_accuracy: 1.0000 - dense_5_accuracy: 1.0000 -
dense_6_accuracy: 1.0000 - dense_7_accuracy: 1.0000 - dense_8_accuracy: 1.0000
```

Out[7]:

```
[0.11270669284309552,
 1.0711692e-10,
 0.11243874,
 3.5251649e-06,
 4.2239775e-05,
 3.9821392e-05,
 6.1821756e-06,
 3.0734265e-07,
 1.0,
 0.973,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0]
```

Задание 2.

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

In [8]:

```
training_dataset = scipy.io.loadmat(os.path.join(data_folder, 'train_32x32.mat'))
X_train, y_train = training_dataset["X"], training_dataset["y"]

X_train = np.moveaxis(X_train, -1, 0)

dev_dataset = scipy.io.loadmat(os.path.join(data_folder, 'test_32x32.mat'))
X_dev, y_dev = dev_dataset["X"], dev_dataset["y"]

X_dev = np.moveaxis(X_dev, -1, 0)
```

In [9]:

```
y_train[y_train == 10] = 0
y_dev[y_dev == 10] = 0

y_train = preprocess(y_train)
y_dev = preprocess(y_dev)
```

In [10]:

```
train(le_net_5_modern_model(X_train.shape[1:]), epochs)
```

```
Train on 73257 samples
Epoch 1/5
73257/73257 [==============================] - 45s 609us/sample - loss: 3.6116 - dense_11_loss: 0.
2022 - dense_12_loss: 2.6632 - dense_13_loss: 0.1605 - dense_14_loss: 0.2222 - dense_15_loss: 0.13
30 - dense_16_loss: 0.1603 - dense_17_loss: 0.0644 - dense_11_accuracy: 0.9856 -
dense_12_accuracy: 0.3099 - dense_13_accuracy: 0.9868 - dense_14_accuracy: 0.9862 -
dense_15_accuracy: 0.9948 - dense_16_accuracy: 0.9860 - dense_17_accuracy: 0.9900
Epoch 2/5
73257/73257 [==============================] - 43s 593us/sample - loss: 1.0601 - dense_11_loss: 2.
2654e-04 - dense_12_loss: 1.0586 - dense_13_loss: 4.4827e-04 - dense_14_loss: 1.7930e-04 - dense_1
5_loss: 5.0013e-05 - dense_16_loss: 1.8471e-04 - dense_17_loss: 2.9696e-04 - dense_11_accuracy: 1.
0000 - dense_12_accuracy: 0.6764 - dense_13_accuracy: 1.0000 - dense_14_accuracy: 1.0000 -
dense_15_accuracy: 1.0000 - dense_16_accuracy: 1.0000 - dense_17_accuracy: 1.0000
Epoch 3/5
73257/73257 [==============================] - 41s 562us/sample - loss: 0.7572 - dense_11_loss: 9.
8369e-05 - dense_12_loss: 0.7567 - dense_13_loss: 2.2222e-04 - dense_14_loss: 1.2206e-04 - dense_1
5_loss: 2.5367e-05 - dense_16_loss: 1.1776e-04 - dense_17_loss: 1.7715e-04 - dense_11_accuracy: 1.
0000 - dense_12_accuracy: 0.7783 - dense_13_accuracy: 1.0000 - dense_14_accuracy: 1.0000 -
dense_15_accuracy: 1.0000 - dense_16_accuracy: 1.0000 - dense_17_accuracy: 1.0000
Epoch 4/5
73257/73257 [==============================] - 41s 561us/sample - loss: 0.6482 - dense_11_loss: 7.
4954e-05 - dense_12_loss: 0.6472 - dense_13_loss: 1.8383e-04 - dense_14_loss: 9.9375e-05 - dense_1
```

```
5_loss: 3.0684e-05 - dense_16_loss: 7.1611e-05 - dense_17_loss: 1.0096e-04 - dense_11_accuracy: 1.
0000 - dense_12_accuracy: 0.8129 - dense_13_accuracy: 1.0000 - dense_14_accuracy: 1.0000 -
dense_15_accuracy: 1.0000 - dense_16_accuracy: 1.0000 - dense_17_accuracy: 1.0000
Epoch 5/5
73257/73257 [==============================] - 41s 566us/sample - loss: 0.5944 - dense_11_loss: 8.
5858e-05 - dense_12_loss: 0.5942 - dense_13_loss: 1.6579e-04 - dense_14_loss: 1.0570e-04 - dense_1
5_loss: 3.2646e-05 - dense_16_loss: 6.0651e-05 - dense_17_loss: 9.2520e-05 - dense_11_accuracy: 1.
0000 - dense_12_accuracy: 0.8287 - dense_13_accuracy: 1.0000 - dense_14_accuracy: 1.0000 -
dense_15_accuracy: 1.0000 - dense_16_accuracy: 1.0000 - dense_17_accuracy: 1.0000
26032/26032 [==============================] - 9s 355us/sample - loss: 0.7215 - dense_11_loss: 1.0
190e-04 - dense_12_loss: 0.7209 - dense_13_loss: 7.2816e-05 - dense_14_loss: 7.1307e-05 - dense_15
_loss: 2.0465e-05 - dense_16_loss: 4.2302e-05 - dense_17_loss: 9.6373e-05 - dense_11_accuracy: 1.0
000 - dense_12_accuracy: 0.8019 - dense_13_accuracy: 1.0000 - dense_14_accuracy: 1.0000 -
dense_15_accuracy: 1.0000 - dense_16_accuracy: 1.0000 - dense_17_accuracy: 1.0000
```

Out[10]:

```
[0.7214683724662315,
 0.00010190105,
 0.72090775,
 7.2816394e-05,
 7.1306946e-05,
 2.0465208e-05,
 4.230245e-05,
 9.6373355e-05,
 1.0,
 0.8019361,
 1.0,
 1.0,
 1.0,
 1.0,
 1.0]
```

In [ ]:

```
def parse_digit_struct(path):
    with h5py.File(path, 'r') as f:
        digit_struct = f['digitStruct']
        bboxes = digit_struct['bbox']
        names = digit_struct['name']

        y = []

        for i in range(len(bboxes)):
            bbox = bboxes[i].item()
            label = f[bbox]['label']

            y_i = [int(label[0][0])] if len(label) <= 1 else [int(f[label[j].item()][0][0])
                                                              for j in range(len(label))]
            if len(y_i) > max_len:
                print('Increase max len up to {}', len(y_i))
                continue

            y_i = int(''.join(map(str, [0 if digit == 10 else digit for digit in y_i])))

            y.append(y_i)

        return np.array(y).reshape((len(y), 1))
```

In [12]:

```
y_train = parse_digit_struct(os.path.join(data_folder, 'train', 'digitStruct.mat'))
y_dev = parse_digit_struct(os.path.join(data_folder, 'test', 'digitStruct.mat'))

y_train = preprocess(y_train)
y_dev = preprocess(y_dev)
```

In [13]:

```
def load_data(name, h , w):
    X = []

    path = os.path.join(data_folder, name)
```

```
    files = sorted(
        filter(
            lambda file_name: file_name.endswith('.png'),
            os.listdir(path)),
        key=lambda file_name: int(file_name[: -len('.png')]))

    for file_name in files:
        try:
            img_path = os.path.join(path, file_name)
            img = Image.open(img_path)
            img = img.resize((h, w))
            img = np.array(img)

            X.append(img)
        except:
            pass

    return np.array(X)
```

In [14]:

```
X_train = load_data('train', 128, 128)
X_dev = load_data('test', 128, 128)
```

In [15]:

```
train(le_net_5_modern_model(X_train.shape[1:]), epochs)
```

```
Train on 33402 samples
Epoch 1/5
33402/33402 [==============================] - 99s 3ms/sample - loss: 19.1338 - dense_20_loss: 3.5
059 - dense_21_loss: 5.0858 - dense_22_loss: 5.1958 - dense_23_loss: 2.7071 - dense_24_loss: 0.940
8 - dense_25_loss: 0.7455 - dense_26_loss: 0.9054 - dense_20_accuracy: 0.5045 - dense_21_accuracy:
0.2287 - dense_22_accuracy: 0.1359 - dense_23_accuracy: 0.6524 - dense_24_accuracy: 0.9422 -
dense_25_accuracy: 0.9837 - dense_26_accuracy: 0.9690
Epoch 2/5
33402/33402 [==============================] - 97s 3ms/sample - loss: 6.7327 - dense_20_loss: 0.91
94 - dense_21_loss: 1.9993 - dense_22_loss: 2.2835 - dense_23_loss: 1.2363 - dense_24_loss: 0.2807
- dense_25_loss: 0.0052 - dense_26_loss: 0.0053 - dense_20_accuracy: 0.6205 - dense_21_accuracy: 0
.3014 - dense_22_accuracy: 0.2168 - dense_23_accuracy: 0.6954 - dense_24_accuracy: 0.9568 -
dense_25_accuracy: 0.9997 - dense_26_accuracy: 1.0000
Epoch 3/5
33402/33402 [==============================] - 98s 3ms/sample - loss: 5.9529 - dense_20_loss: 0.72
35 - dense_21_loss: 1.8027 - dense_22_loss: 2.0580 - dense_23_loss: 1.1218 - dense_24_loss: 0.2402
- dense_25_loss: 0.0040 - dense_26_loss: 0.0024 - dense_20_accuracy: 0.7097 - dense_21_accuracy: 0
.3909 - dense_22_accuracy: 0.3103 - dense_23_accuracy: 0.6967 - dense_24_accuracy: 0.9567 -
dense_25_accuracy: 0.9997 - dense_26_accuracy: 1.0000
Epoch 4/5
33402/33402 [==============================] - 98s 3ms/sample - loss: 5.3776 - dense_20_loss: 0.63
31 - dense_21_loss: 1.6157 - dense_22_loss: 1.8602 - dense_23_loss: 1.0437 - dense_24_loss: 0.2197
- dense_25_loss: 0.0036 - dense_26_loss: 0.0014 - dense_20_accuracy: 0.7453 - dense_21_accuracy: 0
.4587 - dense_22_accuracy: 0.3827 - dense_23_accuracy: 0.7058 - dense_24_accuracy: 0.9567 -
dense_25_accuracy: 0.9997 - dense_26_accuracy: 1.0000
Epoch 5/5
33402/33402 [==============================] - 99s 3ms/sample - loss: 4.9021 - dense_20_loss: 0.56
24 - dense_21_loss: 1.4673 - dense_22_loss: 1.7005 - dense_23_loss: 0.9661 - dense_24_loss: 0.2022
- dense_25_loss: 0.0031 - dense_26_loss: 8.7435e-04 - dense_20_accuracy: 0.7782 -
dense_21_accuracy: 0.5111 - dense_22_accuracy: 0.4430 - dense_23_accuracy: 0.7177 -
dense_24_accuracy: 0.9571 - dense_25_accuracy: 0.9997 - dense_26_accuracy: 1.0000
13068/13068 [==============================] - 32s 2ms/sample - loss: 7.3537 - dense_20_loss: 1.22
30 - dense_21_loss: 2.2428 - dense_22_loss: 2.6212 - dense_23_loss: 1.1236 - dense_24_loss: 0.1419
- dense_25_loss: 0.0031 - dense_26_loss: 7.3890e-04 - dense_20_accuracy: 0.5716 -
dense_21_accuracy: 0.3401 - dense_22_accuracy: 0.2533 - dense_23_accuracy: 0.7303 -
dense_24_accuracy: 0.9861 - dense_25_accuracy: 0.9999 - dense_26_accuracy: 1.0000
```

Out[15]:

```
[7.35365374576509,
 1.2230496,
 2.242823,
 2.6211615,
 1.1236367,
 0.14186539,
 0.0031170375,
```

```
0.000738905,
0.57162535,
0.34014386,
0.25329047,
0.7303336,
0.98607284,
0.99992347,
1.0]
```