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

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

```
import os
try:
  import wget
except:
  !pip install wget
  import wget
import tarfile


out_dir = 'data/svhn'

train_32_32 = ('http://ufldl.stanford.edu/housenumbers/train_32x32.mat', 'train_32x32.mat')
test_32_32 = ('http://ufldl.stanford.edu/housenumbers/test_32x32.mat', 'test_32x32.mat')
extra_32_32 = ('http://ufldl.stanford.edu/housenumbers/extra_32x32.mat', 'extra_32x32.mat')

train_large = ('http://ufldl.stanford.edu/housenumbers/train.tar.gz', 'train.tar.gz')
test_large = ('http://ufldl.stanford.edu/housenumbers/test.tar.gz', 'test.tar.gz')
extra_large = ('http://ufldl.stanford.edu/housenumbers/extra.tar.gz', 'extra.tar.gz')
```

```
Collecting wget
  Downloading https://files.pythonhosted.org/packages/47/6a/62e288da7bcda82b935ff0c6cfe542970f04e29c756b0e147251b2fb2!
Building wheels for collected packages: wget
  Building wheel for wget (setup.py) ... done
  Created wheel for wget: filename=wget-3.2-cp36-none-any.whl size=9682 sha256=2d2a25fd20a30431f1d63c65d682e8b46e4bb6:
  Stored in directory: /root/.cache/pip/wheels/40/15/30/7d8f7cea2902b4db79e3fea550d7d7b85ecb27ef992b618f3f
Successfully built wget
Installing collected packages: wget
Successfully installed wget-3.2
```

```
def download_data(url, filename, out_dir=out_dir):
    filename = os.path.join(out_dir, filename)

    if not os.path.exists(out_dir):
        os.makedirs(out_dir)

    if not os.path.exists(filename):
        print(f"Downloading {filename}.")
        wget.download(url, filename)
        print()
    else:
        print(f"Skipping {filename} download (already exists)")


def extract_data(filename, out_dir=out_dir):
    filename = os.path.join(out_dir, filename)

    print(f"Extracting {filename}")
    with tarfile.open(filename) as tar:
        tar.extractall(out_dir)

download_data(*train_32_32)
download_data(*test_32_32)
download_data(*extra_32_32)

download_data(*train_large)
download_data(*test_large)
# download_data(*extra_large)

extract_data(train_large[1])
extract_data(test_large[1])
# extract_data(extra_large[1])
```

```
Downloading data/svhn/train_32x32.mat.

Downloading data/svhn/test_32x32.mat.

Downloading data/svhn/extra_32x32.mat.

Downloading data/svhn/train.tar.gz.

Downloading data/svhn/test.tar.gz.

Extracting data/svhn/train.tar.gz
Extracting data/svhn/test.tar.gz
```

```
from tensorflow import keras
import numpy as np
from PIL import Image
from pathlib import Path
from scipy import io

def to_one_hot(a, n):
    result = np.zeros(shape=(a.shape[0], n))
    result[np.arange(len(a)), a] = 1
    return result

def load_mnist():
    (x_train, y_train), (x_test, y_test) = keras.datasets.mnist.load_data()

    def to_x(a):
        x = np.array([np.array(Image.fromarray(i).resize((32, 32))) for i in a])
        return x.reshape(x.shape + (1,))

    def to_y(a):
        return to_one_hot(a, 10)

    x_train, y_train = to_x(x_train), to_y(y_train)
    x_test, y_test = to_x(x_test), to_y(y_test)
    print('Loaded and processed mnist dataset')
```

```
    return x_train, y_train, x_test, y_test

def load_single_digit_data(dir='data/svhn', extra=False, greyscale=True):

    def to_x(a):
        a = np.array([a[:,:,:,i] for i in range(a.shape[3])])
        if greyscale:
            return np.mean(a, axis=-1, keepdims=True).astype(np.uint8)
        return a

    def to_y(a):
        y = np.copy(a)
        y = y.reshape(y.shape[0])
        y[y == 10] = 0
        return to_one_hot(y, 10)

    def load_file(file):
        cache_file = Path(dir) / f"{file}.cache.npz"
        if cache_file.exists():
            f = np.load(cache_file)
            print(f'Loaded cached arrays for {file}')
            return [v for k, v in f.items()]

        f = io.loadmat(Path(dir) / file)
        x, y = to_x(f['X']), to_y(f['y'])
        np.savez(Path(dir) / f"{file}.cache.npz", x, y)
        print(f'Loaded and processed {file}')
        return x, y
    x_train, y_train = load_file('train_32x32.mat')
    x_test, y_test = load_file('test_32x32.mat')

    x_extra, y_extra = None, None
    if extra:
        x_extra, y_extra = load_file('extra_32x32.mat')

    return (
        x_train, y_train,
        x_test, y_test,
```

```
        x_extra, y_extra
    )

x_train, y_train, x_test1, y_test1 = load_mnist()
_, _, x_test, y_test, _, _ = load_single_digit_data(extra=False)
```

```
Downloading data from https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz
11493376/11490434 [==============================] - 0s 0us/step
Loaded and processed mnist dataset
Loaded and processed train_32x32.mat
Loaded and processed test_32x32.mat
```

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).

Была реализована сверточная сеть и обучена сначала на mnist наборах данных, потом дообучена на на реальных данных.

```
model = keras.Sequential([
            keras.layers.Conv2D(16, 5, activation='relu', input_shape=x_train.shape[1:], padding='same'),
            keras.layers.MaxPool2D(pool_size=(2, 2), padding = 'same'),
            keras.layers.Conv2D(32, 5, activation='relu', padding='same'),
            keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
            keras.layers.Conv2D(64, 5, activation='relu', padding='same'),
            keras.layers.MaxPool2D(pool_size=(2, 2), padding='same'),
            keras.layers.Flatten(),
            keras.layers.Dropout(rate=0.1),
            keras.layers.Dense(100, activation='relu'),
            keras.layers.Dropout(rate=0.1),
            keras.layers.Dense(y_train.shape[1], activation='softmax')
        ])

model.compile(
            optimizer=keras.optimizers.Adam(0.001),
            loss='categorical_crossentropy',
            metrics=['categorical_accuracy']
        )
model.summary()
```

```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #   
=================================================================
conv2d (Conv2D)              (None, 32, 32, 16)        416       
_________________________________________________________________
max_pooling2d (MaxPooling2D) (None, 16, 16, 16)        0         
_________________________________________________________________
conv2d_1 (Conv2D)            (None, 16, 16, 32)        12832     
_________________________________________________________________
max_pooling2d_1 (MaxPooling2 (None, 8, 8, 32)          0         
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 8, 8, 64)          51264     
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 4, 4, 64)          0         
_________________________________________________________________
flatten (Flatten)            (None, 1024)              0         
_________________________________________________________________
dropout (Dropout)            (None, 1024)              0         
_________________________________________________________________
dense (Dense)                (None, 100)               102500    
_________________________________________________________________
dropout_1 (Dropout)          (None, 100)               0         
_________________________________________________________________
dense_1 (Dense)              (None, 10)                1010      
=================================================================
Total params: 168,022
Trainable params: 168,022
Non-trainable params: 0
_________________________________________________________________
```

```
from sklearn.model_selection import train_test_split
x_train, x_val, y_train, y_val = \
        train_test_split(x_train, y_train, test_size=0.1)

model.fit(
                x_train,
                y_train,
                epochs=100,
```

```
        verbose=2,
        batch_size=100,
        validation_data=(x_val, y_val),
        callbacks=[
            keras.callbacks.EarlyStopping(
                patience=10,
                restore_best_weights=True
            )
        ]
    )
```

```
Epoch 1/100
540/540 - 2s - loss: 0.4744 - categorical_accuracy: 0.9110 - val_loss: 0.0631 - val_categorical_accuracy: 0.9807
Epoch 2/100
540/540 - 2s - loss: 0.0706 - categorical_accuracy: 0.9782 - val_loss: 0.0544 - val_categorical_accuracy: 0.9837
Epoch 3/100
540/540 - 2s - loss: 0.0493 - categorical_accuracy: 0.9850 - val_loss: 0.0488 - val_categorical_accuracy: 0.9848
Epoch 4/100
540/540 - 2s - loss: 0.0417 - categorical_accuracy: 0.9874 - val_loss: 0.0405 - val_categorical_accuracy: 0.9875
Epoch 5/100
540/540 - 2s - loss: 0.0394 - categorical_accuracy: 0.9880 - val_loss: 0.0407 - val_categorical_accuracy: 0.9867
Epoch 6/100
540/540 - 2s - loss: 0.0336 - categorical_accuracy: 0.9896 - val_loss: 0.0385 - val_categorical_accuracy: 0.9887
Epoch 7/100
540/540 - 2s - loss: 0.0347 - categorical_accuracy: 0.9892 - val_loss: 0.0361 - val_categorical_accuracy: 0.9905
Epoch 8/100
540/540 - 2s - loss: 0.0272 - categorical_accuracy: 0.9912 - val_loss: 0.0408 - val_categorical_accuracy: 0.9877
Epoch 9/100
540/540 - 2s - loss: 0.0278 - categorical_accuracy: 0.9909 - val_loss: 0.0285 - val_categorical_accuracy: 0.9918
Epoch 10/100
540/540 - 2s - loss: 0.0278 - categorical_accuracy: 0.9914 - val_loss: 0.0331 - val_categorical_accuracy: 0.9898
Epoch 11/100
540/540 - 2s - loss: 0.0240 - categorical_accuracy: 0.9928 - val_loss: 0.0446 - val_categorical_accuracy: 0.9887
Epoch 12/100
540/540 - 2s - loss: 0.0264 - categorical_accuracy: 0.9918 - val_loss: 0.0504 - val_categorical_accuracy: 0.9887
Epoch 13/100
540/540 - 2s - loss: 0.0247 - categorical_accuracy: 0.9929 - val_loss: 0.0498 - val_categorical_accuracy: 0.9858
Epoch 14/100
540/540 - 2s - loss: 0.0228 - categorical_accuracy: 0.9929 - val_loss: 0.0290 - val_categorical_accuracy: 0.9917
Epoch 15/100
540/540 - 2s - loss: 0.0248 - categorical_accuracy: 0.9925 - val_loss: 0.0397 - val_categorical_accuracy: 0.9912
Epoch 16/100
540/540 - 2s - loss: 0.0188 - categorical_accuracy: 0.9939 - val_loss: 0.0422 - val_categorical_accuracy: 0.9910
Epoch 17/100
540/540 - 2s - loss: 0.0218 - categorical_accuracy: 0.9934 - val_loss: 0.0443 - val_categorical_accuracy: 0.9905
Epoch 18/100
540/540 - 2s - loss: 0.0195 - categorical_accuracy: 0.9942 - val_loss: 0.0488 - val_categorical_accuracy: 0.9880
Epoch 19/100
540/540 - 2s - loss: 0.0179 - categorical_accuracy: 0.9949 - val_loss: 0.0439 - val_categorical_accuracy: 0.9897
<tensorflow.python.keras.callbacks.History at 0x7f0e1f7c1748>
```

```
_, acc = model.evaluate(x_test, y_test)
print(f'Accuracy = {acc:.5f}')
```

```
814/814 [==============================] - 2s 2ms/step - loss: 2.2326 - categorical_accuracy: 0.2302
Accuracy = 0.23022
```

```
_, acc = model.evaluate(x_test1, y_test1)
print(f'Accuracy = {acc:.5f}')
```

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0367 - categorical_accuracy: 0.9903
Accuracy = 0.99030
```

```
model.save_weights('models/svhn_mnist_conv_net_svhn/model')
```

```
!ls models/svhn_mnist_conv_net_svhn
```

```
checkpoint  model.data-00000-of-00002  model.data-00001-of-00002  model.index
```

```
x_train, y_train, x_test, y_test, _, _ = load_single_digit_data(extra=False)
model.fit(
                x_train,
                y_train,
                epochs=100,
                verbose=2,
                batch_size=100,
                validation_split=0.1,
                callbacks=[
                    keras.callbacks.EarlyStopping(
                        patience=10,
                        restore_best_weights=True
                    )
                ]
            )
```

```
Loaded cached arrays for train_32x32.mat
Loaded cached arrays for test_32x32.mat
Epoch 1/100
660/660 - 3s - loss: 0.8423 - categorical_accuracy: 0.7376 - val_loss: 0.5252 - val_categorical_accuracy: 0.8363
Epoch 2/100
660/660 - 3s - loss: 0.5199 - categorical_accuracy: 0.8428 - val_loss: 0.4507 - val_categorical_accuracy: 0.8630
Epoch 3/100
660/660 - 3s - loss: 0.4375 - categorical_accuracy: 0.8681 - val_loss: 0.4370 - val_categorical_accuracy: 0.8709
Epoch 4/100
660/660 - 3s - loss: 0.3871 - categorical_accuracy: 0.8820 - val_loss: 0.4078 - val_categorical_accuracy: 0.8754
Epoch 5/100
660/660 - 3s - loss: 0.3574 - categorical_accuracy: 0.8906 - val_loss: 0.4060 - val_categorical_accuracy: 0.8827
Epoch 6/100
660/660 - 3s - loss: 0.3337 - categorical_accuracy: 0.8966 - val_loss: 0.3934 - val_categorical_accuracy: 0.8886
Epoch 7/100
660/660 - 3s - loss: 0.3090 - categorical_accuracy: 0.9050 - val_loss: 0.3850 - val_categorical_accuracy: 0.8909
Epoch 8/100
660/660 - 3s - loss: 0.2935 - categorical_accuracy: 0.9093 - val_loss: 0.3699 - val_categorical_accuracy: 0.8950
Epoch 9/100
660/660 - 3s - loss: 0.2702 - categorical_accuracy: 0.9156 - val_loss: 0.3987 - val_categorical_accuracy: 0.8860
Epoch 10/100
660/660 - 3s - loss: 0.2614 - categorical_accuracy: 0.9185 - val_loss: 0.4142 - val_categorical_accuracy: 0.8840
Epoch 11/100
660/660 - 3s - loss: 0.2480 - categorical_accuracy: 0.9204 - val_loss: 0.3982 - val_categorical_accuracy: 0.8931
Epoch 12/100
660/660 - 3s - loss: 0.2377 - categorical_accuracy: 0.9246 - val_loss: 0.4121 - val_categorical_accuracy: 0.8893
Epoch 13/100
660/660 - 3s - loss: 0.2263 - categorical_accuracy: 0.9285 - val_loss: 0.4151 - val_categorical_accuracy: 0.8923
Epoch 14/100
660/660 - 3s - loss: 0.2199 - categorical_accuracy: 0.9293 - val_loss: 0.4328 - val_categorical_accuracy: 0.8924
Epoch 15/100
660/660 - 3s - loss: 0.2093 - categorical_accuracy: 0.9324 - val_loss: 0.4270 - val_categorical_accuracy: 0.8957
Epoch 16/100
660/660 - 3s - loss: 0.2029 - categorical_accuracy: 0.9352 - val_loss: 0.4190 - val_categorical_accuracy: 0.8894
Epoch 17/100
660/660 - 3s - loss: 0.1972 - categorical_accuracy: 0.9358 - val_loss: 0.4606 - val_categorical_accuracy: 0.8908
Epoch 18/100
660/660 - 3s - loss: 0.1919 - categorical_accuracy: 0.9376 - val_loss: 0.4521 - val_categorical_accuracy: 0.8965
<tensorflow.python.keras.callbacks.History at 0x7f0e1c4bd4a8>
```

```
_, acc = model.evaluate(x_test1, y_test1)
print(f'Accuracy = {acc:.5f}')
_, acc = model.evaluate(x_test, y_test)
print(f'Accuracy = {acc:.5f}')
```

```
313/313 [==============================] - 1s 2ms/step - loss: 2.2770 - categorical_accuracy: 0.6756
Accuracy = 0.67560
814/814 [==============================] - 2s 2ms/step - loss: 0.4020 - categorical_accuracy: 0.8930
Accuracy = 0.89302
```

```
_, _, x_test, y_test, x_extra, y_extra = load_single_digit_data(extra=True)
model.fit(
                x_extra,
                y_extra,
                epochs=100,
                verbose=2,
                batch_size=100,
                validation_split=0.1,
                callbacks=[
                    keras.callbacks.EarlyStopping(
                        patience=10,
                        restore_best_weights=True
                    )
                ]
            )
```

```
Loaded cached arrays for train_32x32.mat
Loaded cached arrays for test_32x32.mat
Loaded and processed extra_32x32.mat
Epoch 1/100
4781/4781 - 19s - loss: 0.2035 - categorical_accuracy: 0.9426 - val_loss: 0.1371 - val_categorical_accuracy: 0.9618
Epoch 2/100
4781/4781 - 19s - loss: 0.1615 - categorical_accuracy: 0.9548 - val_loss: 0.1374 - val_categorical_accuracy: 0.9613
Epoch 3/100
4781/4781 - 19s - loss: 0.1482 - categorical_accuracy: 0.9585 - val_loss: 0.1188 - val_categorical_accuracy: 0.9667
Epoch 4/100
4781/4781 - 19s - loss: 0.1388 - categorical_accuracy: 0.9614 - val_loss: 0.1211 - val_categorical_accuracy: 0.9669
Epoch 5/100
4781/4781 - 19s - loss: 0.1349 - categorical_accuracy: 0.9624 - val_loss: 0.1153 - val_categorical_accuracy: 0.9681
Epoch 6/100
4781/4781 - 19s - loss: 0.1287 - categorical_accuracy: 0.9643 - val_loss: 0.1143 - val_categorical_accuracy: 0.9690
Epoch 7/100
4781/4781 - 18s - loss: 0.1247 - categorical_accuracy: 0.9653 - val_loss: 0.1106 - val_categorical_accuracy: 0.9699
Epoch 8/100
4781/4781 - 19s - loss: 0.1225 - categorical_accuracy: 0.9659 - val_loss: 0.1194 - val_categorical_accuracy: 0.9690
Epoch 9/100
4781/4781 - 19s - loss: 0.1196 - categorical_accuracy: 0.9665 - val_loss: 0.1194 - val_categorical_accuracy: 0.9669
Epoch 10/100
4781/4781 - 19s - loss: 0.1187 - categorical_accuracy: 0.9670 - val_loss: 0.1091 - val_categorical_accuracy: 0.9715
Epoch 11/100
4781/4781 - 19s - loss: 0.1166 - categorical_accuracy: 0.9676 - val_loss: 0.1161 - val_categorical_accuracy: 0.9694
Epoch 12/100
4781/4781 - 19s - loss: 0.1137 - categorical_accuracy: 0.9686 - val_loss: 0.1087 - val_categorical_accuracy: 0.9713
Epoch 13/100
4781/4781 - 19s - loss: 0.1132 - categorical_accuracy: 0.9684 - val_loss: 0.1206 - val_categorical_accuracy: 0.9680
Epoch 14/100
4781/4781 - 19s - loss: 0.1129 - categorical_accuracy: 0.9684 - val_loss: 0.1103 - val_categorical_accuracy: 0.9711
Epoch 15/100
4781/4781 - 19s - loss: 0.1105 - categorical_accuracy: 0.9695 - val_loss: 0.1217 - val_categorical_accuracy: 0.9710
Epoch 16/100
4781/4781 - 19s - loss: 0.1117 - categorical_accuracy: 0.9691 - val_loss: 0.1150 - val_categorical_accuracy: 0.9715
Epoch 17/100
4781/4781 - 19s - loss: 0.1094 - categorical_accuracy: 0.9697 - val_loss: 0.1221 - val_categorical_accuracy: 0.9686
Epoch 18/100
4781/4781 - 19s - loss: 0.1097 - categorical_accuracy: 0.9697 - val_loss: 0.1202 - val_categorical_accuracy: 0.9693
Epoch 19/100
4781/4781 - 18s - loss: 0.1081 - categorical_accuracy: 0.9700 - val_loss: 0.1335 - val_categorical_accuracy: 0.9637
Epoch 20/100
```

```
4781/4781 - 18s - loss: 0.1088 - categorical_accuracy: 0.9699 - val_loss: 0.1264 - val_categorical_accuracy: 0.9703
Epoch 21/100
4781/4781 - 19s - loss: 0.1059 - categorical_accuracy: 0.9707 - val_loss: 0.1285 - val_categorical_accuracy: 0.9662
Epoch 22/100
4781/4781 - 19s - loss: 0.1072 - categorical_accuracy: 0.9704 - val_loss: 0.1223 - val_categorical_accuracy: 0.9701
<tensorflow.python.keras.callbacks.History at 0x7f0e1c2d7748>
```

```
_, acc = model.evaluate(x_test1, y_test1)
print(f'Accuracy = {acc:.5f}')
_, acc = model.evaluate(x_test, y_test)
print(f'Accuracy = {acc:.5f}')
```

```
313/313 [==============================] - 1s 2ms/step - loss: 49.9188 - categorical_accuracy: 0.2609
Accuracy = 0.26090
814/814 [==============================] - 2s 2ms/step - loss: 0.2932 - categorical_accuracy: 0.9264
Accuracy = 0.92644
```

```
import tensorflow as tf
tf.test.gpu_device_name()
```

```
'/device:GPU:0'
```