

Лабораторная работа №1. Логистическая регрессия в качестве нейронной сети

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A ... J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке: https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz

(https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz) (большой набор данных);

https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz

(https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz) (маленький набор данных); Описание данных на

английском языке доступно по ссылке: <http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html>

(<http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html>)

```
In [0]: 1 import numpy as np
        2 import pandas as pd
        3 import matplotlib.pyplot as plt
        4 import matplotlib.image as mpimg
        5 from scipy import misc
        6 import glob
        7 import hashlib
        8 from tqdm import tqdm
        9 from sklearn.linear_model import LogisticRegression
```

Задание 1. Загрузите данные и отобразите на экране несколько из изображений с помощью языка Python;

```
In [0]: 1 def plot(imagesDict):
2         _, axis = plt.subplots(1, len(imagesDict))
3         count = 0
4
5         for key in imagesDict.keys():
6             img = mpimg.imread(list(imagesDict[key])[0])
7             axis[count].imshow(img)
8             axis[count].axis("off")
9             count += 1
10
11         plt.show()
12
```

```
In [0]: 1 def md5(fname):
2         hash_md5 = hashlib.md5()
3         with open(fname, "rb") as f:
4             for chunk in iter(lambda: f.read(4096), b''):
5                 hash_md5.update(chunk)
6         return hash_md5.hexdigest()
7
8 def removeDublicates(dataset):
9     result = dict()
10    for file in dataset:
11        result[md5(file)] = file
12    return list(result.values())
```

```
In [0]: 1 rootFolderName = "notMNIST_large/"
        2 folders = glob.glob(rootFolderName + "*")
        3
        4 imagesDict = dict()
        5 for folder in folders:
        6     images = glob.glob(folder + "/*.png")
        7     imagesSet = list()
        8     for image in images:
        9         imagesSet.append(image)
       10     imagesSet = removeDublicates(imagesSet)
       11     imagesDict[folder.replace(rootFolderName, "")] = imagesSet
```

```
In [0]: 1 plot(imagesDict)
```



```
In [0]: 1 keys = imagesDict.keys()
        2 print(keys)
```

```
dict_keys(['I', 'G', 'A', 'F', 'H', 'J', 'C', 'D', 'E', 'B'])
```

Задание 2. Проверьте, что классы являются сбалансированными, т.е. количество изображений, принадлежащих каждому из классов, примерно одинаково (В данной задаче 10 классов).

```
In [0]: 1 for key in keys:
        2     print(key, len(imagesDict[key]))
```

```
I 41173
G 47090
A 47107
F 46844
H 46183
J 46658
C 46654
D 46734
E 46954
B 47283
```

Задание 3. Разделите данные на три подвыборки: обучающую (200 тыс. изображений), валидационную (10 тыс. изображений) и контрольную (тестовую) (19 тыс. изображений);

```
In [0]: 1 trainDict = dict()
        2 validationDict = dict()
        3 testDict = dict()
        4
        5 matches = {"A":0, "B":1, "C":2, "D":3, "E":4, "F":5, "G":6, "H":7, "I":8, "J":9}
        6
        7 for key in imagesDict.keys():
        8     imagesList = imagesDict[key]
        9     newKey = matches[key]
       10     trainDict[newKey] = imagesList[:20000]
       11     validationDict[newKey] = imagesList[20000:21000]
       12     testDict[newKey] = imagesList[21000:23000]
```

Задание 4. Проверьте, что данные из обучающей выборки не пересекаются с данными из валидационной и контрольной выборок. Другими словами, избавьтесь от дубликатов в обучающей выборке.

```
In [0]: 1 for key in trainDict.keys():
        2     trainSet = set(trainDict[key])
        3     validationSet = set(validationDict[key])
        4     testSet = set(testDict[key])
        5     intersactionWithValidation = trainSet.intersection(validationSet)
        6     intersactionWithTest = trainSet.intersection(testSet)
        7     if len(intersactionWithValidation) > 0 or len(intersactionWithTest) > 0:
        8         print("Warning: sets intersaction")
        9
```


Задание 5. Постройте простейший классификатор (например, с помощью логистической регрессии). Постройте график зависимости точности классификатора от размера обучающей выборки (50, 100, 1000, 50000). Для построения классификатора можете использовать библиотеку SkLearn (<http://scikit-learn.org> (<http://scikit-learn.org>)).


```
In [0]: 1 def convertToLearnData(dataset):
2       y = np.zeros(0)
3       x = np.zeros(0)
4
5       for key in tqdm(dataset):
6           for path in dataset[key]:
7               try:
8                   image = mpimg.imread(path)
9               except:
10                  print(path)
11                  image = image.reshape(1, 784)
12                  if len(x) > 0 and len(y) > 0:
13                      x = np.append(x, image, axis=0)
14                      y = np.append(y, key)
15                  else:
16                      x = image
17                      y = np.array([key])
18       return x, y
```


```
In [0]: 1 trainX, trainY = convertToLearnData(trainDict)
2       print("Train:", trainX.shape, trainY.shape)
```


0% | 0/10 [00:00<?, ?it/s]


10% | 1/10 [06:06<54:57, 366.44s/it]


20% |  | 2/10 [26:14<1:22:32, 619.01s/it]


30% |  | 3/10 [51:41<1:43:59, 891.30s/it]


40% |  | 4/10 [1:25:02<2:02:25, 1224.32s/it]

50% |  | 5/10 [2:07:08<2:14:33, 1614.73s/it]

60% |  | 6/10 [2:59:14<2:17:52, 2068.22s/it]

70% |  | 7/10 [4:00:48<2:07:47, 2555.94s/it]

80% |  | 8/10 [5:14:36<1:43:54, 3117.40s/it]

90% |  | 9/10 [6:36:48<1:01:01, 3661.78s/it]

100%|██████████| 10/10 [8:04:41<00:00, 2908.14s/it]

Train: (200000, 784) (200000,)

```
In [0]: 1 validationX, validationY = convertToLearnData(validationDict)
        2 print("Validation:", validationX.shape, validationY.shape)
        3
```

0%| | 0/10 [00:00<?, ?it/s]

10%|█ | 1/10 [00:01<00:16, 1.78s/it]

20%|██ | 2/10 [00:05<00:18, 2.35s/it]

30%|███ | 3/10 [00:10<00:22, 3.23s/it]

40%|████ | 4/10 [00:17<00:25, 4.26s/it]

50%|█████ | 5/10 [00:25<00:27, 5.54s/it]

60%|██████ | 6/10 [00:35<00:27, 6.88s/it]

70%|██████ | 7/10 [00:47<00:24, 8.29s/it]

80%|██████ | 8/10 [01:00<00:19, 9.74s/it]


90%|██████ | 9/10 [01:15<00:11, 11.27s/it]


100%|██████████| 10/10 [01:31<00:00, 9.19s/it]


Validation: (10000, 784) (10000,)


```
In [0]: 1 testX, testY = convertToLearnData(testDict)
        2 print("Test:", testX.shape, testY.shape)
```


0%| | 0/10 [00:00<?, ?it/s]


10% |  | 1/10 [00:04<00:44, 4.97s/it]


20% |  | 2/10 [00:16<00:55, 6.89s/it]


30% |  | 3/10 [00:34<01:11, 10.25s/it]


40% |  | 4/10 [00:58<01:27, 14.54s/it]

50% |  | 5/10 [01:30<01:37, 19.59s/it]

60% |  | 6/10 [02:08<01:40, 25.08s/it]

70% |  | 7/10 [02:52<01:32, 30.90s/it]

80% |  | 8/10 [03:43<01:13, 36.87s/it]

90% |  | 9/10 [04:40<00:42, 42.90s/it]

100%|██████████| 10/10 [05:44<00:00, 34.41s/it]

Test: (20000, 784) (20000,)

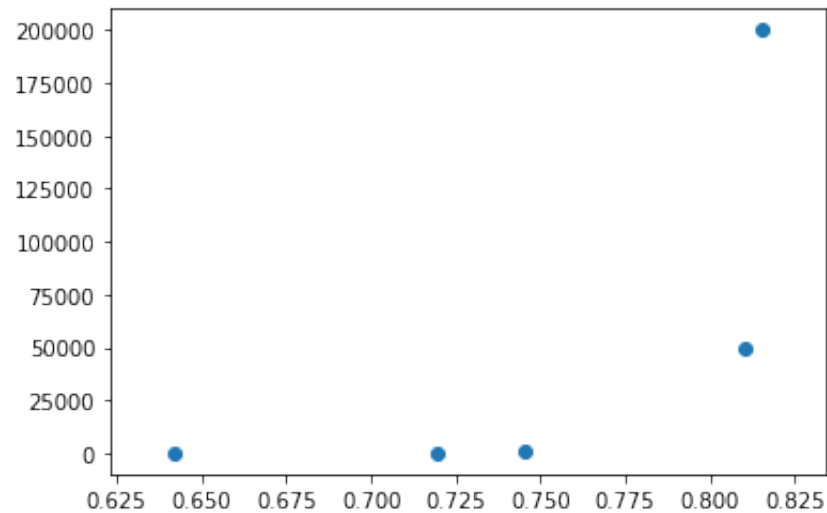
```
In [0]: 1 trainSizes = [50, 100, 1000, 50000]
        2
        3 learnDictData = dict()
        4
        5 for size in trainSizes:
        6     newDict = dict()
        7     sliceSize = int(size / 10)
        8     for key in trainDict.keys():
        9         newDict[key] = trainDict[key][:sliceSize]
       10     newTrainX, newTrainY = convertToLearnData(newDict)
       11     clf = LogisticRegression(random_state=0).fit(newTrainX, newTrainY)
       12     score = clf.score(validationX, validationY)
       13     learnDictData[size] = score
       14
```

```
In [0]: 1 clf = LogisticRegression(random_state=0).fit(trainX, trainY)
        2 learnDictData[200000] = clf.score(validationX, validationY)
```

```
In [0]: 1 learnDictData
```

```
Out[136]: {50: 0.6416, 100: 0.7194, 1000: 0.7453, 50000: 0.8101, 200000: 0.8156}
```

```
In [0]: 1 iterations = learnDictData.keys()  
2 scores = learnDictData.values()  
3  
4 plt.scatter(scores, iterations)  
5 plt.show()
```



Лабораторная работа №2. Реализация глубокой нейронной сети

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A ... J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке:

https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz

(https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz) (большой набор данных);

https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz

(https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz) (маленький набор данных);

Описание данных на английском языке доступно по ссылке: <http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html>

(<http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html>)

In [0]:

```
1 import tensorflow as tf
2 from tensorflow import keras
3 import numpy as np
4 import matplotlib.pyplot as plt
5
6 from scipy.io import savemat
7 import numpy, glob, sys, os
8 from PIL import Image
9 from scipy.io import loadmat
```

```
In [0]: 1 def generateDataset(folder, target):
2         max_count = 0
3         for (root, dirs, files) in os.walk(folder):
4             for f in files:
5                 if f.endswith('.png'):
6                     max_count += 1
7         print('Found %s files' % (max_count,))
8         data = numpy.zeros((28,28,max_count))
9         labels = numpy.zeros((max_count,))
10        count = 0
11        for (root, dirs, files) in os.walk(folder):
12            for f in files:
13                if f.endswith('.png'):
14                    try:
15                        img = Image.open(os.path.join(root,f));
16                        data[:, :, count]=numpy.asarray(img)
17                        surround_folder = os.path.split(root)[-1]
18                        assert len(surround_folder)==1
19                        labels[count]=ord(surround_folder)-ord('A')
20                        count+=1
21                    except:
22                        pass
23
24        print('Saving to ', target)
25        savemat(target,{'images': data[:, :, :count], 'labels': labels[:count]})
```

```
In [0]: 1 generateDataset("notMNIST_small", "test_dataset.mat")
        2 generateDataset("notMNIST_large", "train_dataset.mat")
```

```
Found 18726 files
Saving to test_dataset.mat
Found 529119 files
Saving to train_dataset.mat
```

```
In [0]: 1 def getDataset(file):
        2     matfile = loadmat(file)
        3     x = matfile['images'] / 255
        4     y = matfile['labels']
        5     return x.T, y.T
        6
        7 trainX, trainY = getDataset('train_dataset.mat')
        8 print(trainX.shape, trainY.shape)
        9
       10 testX, testY = getDataset('test_dataset.mat')
       11 print(testX.shape, testY.shape)
```

```
(529115, 28, 28) (529115, 1)
(18724, 28, 28) (18724, 1)
```

Задание 1. Реализуйте полносвязную нейронную сеть с помощью библиотеки Tensor Flow. В качестве алгоритма оптимизации можно использовать, например, стохастический градиент (Stochastic Gradient Descent, SGD). Определите количество скрытых слоев от 1 до 5, количество нейронов в каждом из слоев до нескольких сотен, а также их функции активации (кусочно-линейная, сигмоидная, гиперболический тангенс и т.д.).

```
In [0]: 1 model = keras.Sequential([
        2     keras.layers.Flatten(input_shape=(28, 28)),
        3     keras.layers.Dense(256, activation='relu'),
```

```
4     keras.layers.Dense(128, activation='relu'),
5     keras.layers.Dense(64, activation='relu'),
6     keras.layers.Dense(10, activation='softmax')
7 ])
8
9 model.compile(optimizer='adam',
10               loss='sparse_categorical_crossentropy',
11               metrics=['accuracy'])
12
13 model.fit(trainX, trainY, epochs=15)
```

WARNING:tensorflow:From /Users/alinadolmatovich/anaconda2/envs/tensorflow-env/lib/python3.6/site-packages/tensorflow_core/python/ops/resource_variable_ops.py:1630: calling BaseResourceVariable.__init__ (from tensorflow.python.ops.resource_variable_ops) with constraint is deprecated and will be removed in a future version.

Instructions for updating:

If using Keras pass *_constraint arguments to layers.

Train on 529115 samples

Epoch 1/15

529115/529115 [=====] - 199s 375us/sample - loss: 0.4140 - acc: 0.8734

Epoch 2/15

529115/529115 [=====] - 192s 362us/sample - loss: 0.3290 - acc: 0.8979

Epoch 3/15

529115/529115 [=====] - 190s 360us/sample - loss: 0.3008 - acc: 0.9061

Epoch 4/15

529115/529115 [=====] - 185s 349us/sample - loss: 0.2836 - acc: 0.9110

Epoch 5/15

529115/529115 [=====] - 178s 337us/sample - loss: 0.2704 - acc: 0.9148

Epoch 6/15

529115/529115 [=====] - 179s 338us/sample - loss: 0.2609 - acc: 0.9177

Epoch 7/15

529115/529115 [=====] - 174s 328us/sample - loss: 0.2534 - acc: 0.9199

Epoch 8/15


```

529115/529115 [=====] - 181s 342us/sample - loss: 0.2470 - acc: 0.9218
Epoch 9/15
529115/529115 [=====] - 168s 318us/sample - loss: 0.2403 - acc: 0.9236
Epoch 10/15
529115/529115 [=====] - 191s 362us/sample - loss: 0.2366 - acc: 0.9248
Epoch 11/15
529115/529115 [=====] - 202s 382us/sample - loss: 0.2324 - acc: 0.9259
Epoch 12/15
529115/529115 [=====] - 179s 337us/sample - loss: 0.2280 - acc: 0.9273
Epoch 13/15
529115/529115 [=====] - 187s 353us/sample - loss: 0.2246 - acc: 0.9287
Epoch 14/15
529115/529115 [=====] - 188s 356us/sample - loss: 0.2213 - acc: 0.9293
Epoch 15/15
529115/529115 [=====] - 181s 342us/sample - loss: 0.2184 - acc: 0.9304

```

Out[5]: <tensorflow.python.keras.callbacks.History at 0x13face4a8>

Задание 2. Как улучшилась точность классификатора по сравнению с логистической регрессией?

```

In [0]: 1 testLoss, testAcc = model.evaluate(testX, testY, verbose=2)
        2 print('Точность на проверочных данных:', testAcc)

```

```

18724/18724 - 2s - loss: 0.1378 - acc: 0.9621
Точность на проверочных данных: 0.9621342

```

Задание 3. Используйте регуляризацию и метод сброса нейронов (dropout) для борьбы с переобучением. Как улучшилось качество классификации?

```

In [0]: 1 regularizationModel = keras.models.Sequential([
        2     keras.layers.Flatten(input_shape=(28, 28)),
        3     keras.layers.Dense(256, kernel_regularizer=keras.regularizers.l2(0.001), activation='relu'),
        4     keras.layers.Dense(128, kernel_regularizer=keras.regularizers.l2(0.001), activation='relu'),
        5     keras.layers.Dense(64, kernel_regularizer=keras.regularizers.l2(0.001), activation='relu')

```

```
5     keras.layers.Dense(64, kernel_regularizer=keras.regularizers.l2(0.001), activation='relu'),
6     keras.layers.Dense(10, activation='softmax')
7 ])
8
9 regularizationModel.compile(optimizer='adam',
10                             loss='sparse_categorical_crossentropy',
11                             metrics=['accuracy'])
12
13 regularizationModel.fit(trainX, trainY, epochs=15)
```

Train on 529115 samples

Epoch 1/15

529115/529115 [=====] - 251s 475us/sample - loss: 0.6194 - acc: 0.8538

Epoch 2/15

529115/529115 [=====] - 264s 499us/sample - loss: 0.5291 - acc: 0.8673

Epoch 3/15

529115/529115 [=====] - 295s 558us/sample - loss: 0.5174 - acc: 0.8694

Epoch 4/15

529115/529115 [=====] - 306s 579us/sample - loss: 0.5108 - acc: 0.8707

Epoch 5/15

529115/529115 [=====] - 291s 550us/sample - loss: 0.5062 - acc: 0.8709

Epoch 6/15

529115/529115 [=====] - 312s 589us/sample - loss: 0.5040 - acc: 0.8715

Epoch 7/15

529115/529115 [=====] - 295s 557us/sample - loss: 0.5038 - acc: 0.8713

Epoch 8/15

529115/529115 [=====] - 283s 536us/sample - loss: 0.5027 - acc: 0.8714

Epoch 9/15

529115/529115 [=====] - 244s 460us/sample - loss: 0.5017 - acc: 0.8715

Epoch 10/15

529115/529115 [=====] - 239s 452us/sample - loss: 0.4999 - acc: 0.8710

Epoch 11/15

529115/529115 [=====] - 244s 461us/sample - loss: 0.4994 - acc: 0.8715

Epoch 12/15

```
529115/529115 [=====] - 239s 451us/sample - loss: 0.4991 - acc: 0.8710
Epoch 13/15
529115/529115 [=====] - 243s 459us/sample - loss: 0.4976 - acc: 0.8716
Epoch 14/15
529115/529115 [=====] - 251s 475us/sample - loss: 0.4980 - acc: 0.8716
Epoch 15/15
529115/529115 [=====] - 249s 470us/sample - loss: 0.4981 - acc: 0.8712
```

Out[9]: <tensorflow.python.keras.callbacks.History at 0x65c8dde10>

```
In [0]: 1 testRegularizationLoss, testRegularizationAcc = regularizationModel.evaluate(testX, testY, verbose=2)
        2 print('Точность на проверочных данных с регуляризацией:', testRegularizationAcc)
```

```
18724/18724 - 3s - loss: 0.3101 - acc: 0.9301
Точность на проверочных данных с регуляризацией: 0.9300897
```

```
In [0]: 1 dropoutModel = keras.Sequential([
        2     keras.layers.Flatten(input_shape=(28, 28)),
        3     keras.layers.Dropout(0.5),
        4     keras.layers.Dense(256, activation='relu'),
        5     keras.layers.Dropout(0.5),
        6     keras.layers.Dense(128, activation='relu'),
        7     keras.layers.Dropout(0.5),
        8     keras.layers.Dense(64, activation='relu'),
        9     keras.layers.Dropout(0.5),
       10     keras.layers.Dense(10, activation='softmax')
       11 ])
       12
       13 dropoutModel.compile(optimizer='adam',
       14                       loss='sparse_categorical_crossentropy',
       15                       metrics=['accuracy'])
       16
       17 dropoutModel.fit(trainX, trainY, epochs=15)
```

```
Train on 529115 samples
```

```
Epoch 1/15
```

```
529115/529115 [=====] - 284s 536us/sample - loss: 0.7617 - acc: 0.7801
```

```
Epoch 2/15
```

```
529115/529115 [=====] - 284s 537us/sample - loss: 0.6313 - acc: 0.8222
```

```
Epoch 3/15
```

```
529115/529115 [=====] - 280s 529us/sample - loss: 0.6077 - acc: 0.8285
```

```
Epoch 4/15
```

```
529115/529115 [=====] - 280s 529us/sample - loss: 0.5949 - acc: 0.8326
```

```
Epoch 5/15
```

```
529115/529115 [=====] - 287s 542us/sample - loss: 0.5863 - acc: 0.8340
```

```
Epoch 6/15
```

```
529115/529115 [=====] - 279s 527us/sample - loss: 0.5816 - acc: 0.8358
```

```
Epoch 7/15
```

```
529115/529115 [=====] - 306s 579us/sample - loss: 0.5772 - acc: 0.8367
```

```
Epoch 8/15
```

```
529115/529115 [=====] - 284s 537us/sample - loss: 0.5737 - acc: 0.8382
```

```
Epoch 9/15
```

```
529115/529115 [=====] - 314s 593us/sample - loss: 0.5706 - acc: 0.8389
```

```
Epoch 10/15
```

```
529115/529115 [=====] - 365s 691us/sample - loss: 0.5695 - acc: 0.8391
```

```
Epoch 11/15
```

```
529115/529115 [=====] - 381s 720us/sample - loss: 0.5662 - acc: 0.8403
```

```
Epoch 12/15
```

```
529115/529115 [=====] - 435s 822us/sample - loss: 0.5639 - acc: 0.8414- lo
```

```
Epoch 13/15
```

```
529115/529115 [=====] - 406s 767us/sample - loss: 0.5646 - acc: 0.8411
```

```
Epoch 14/15
```

```
529115/529115 [=====] - 325s 615us/sample - loss: 0.5609 - acc: 0.8420
```

```
Epoch 15/15
```

```
529115/529115 [=====] - 344s 650us/sample - loss: 0.5606 - acc: 0.8418
```

```
Out[12]: <tensorflow.python.keras.callbacks.History at 0x65c884fd0>
```

In [0]:

```
1 testDropoutLoss, testDropoutAcc = dropoutModel.evaluate(testX, testY, verbose=2)
2 print('Точность на проверочных данных с dropout\''ом:', testDropoutAcc)
```

18724/18724 - 4s - loss: 0.2280 - acc: 0.9383
Точность на проверочных данных с dropout\''ом: 0.93826103

Задание 4. Воспользуйтесь динамически изменяемой скоростью обучения (learning rate). Наилучшая точность, достигнутая с помощью данной модели составляет 97.1%. Какую точность демонстрирует Ваша реализованная модель?

In [0]:

```
1 model = keras.Sequential([
2     keras.layers.Flatten(input_shape=(28, 28)),
3     keras.layers.Dense(256, activation='relu'),
4     keras.layers.Dense(128, activation='relu'),
5     keras.layers.Dense(64, activation='relu'),
6     keras.layers.Dense(10, activation='softmax')
7 ])
8
9 adamOptimizer = keras.optimizers.Adam(learning_rate=0.01)
10 model.compile(optimizer=adamOptimizer,
11               loss='sparse_categorical_crossentropy',
12               metrics=['accuracy'])
13
14 model.fit(trainX, trainY, epochs=15)
```

Train on 529115 samples

Epoch 1/15

529115/529115 [=====] - 238s 450us/sample - loss: 0.5622 - acc: 0.8365

Epoch 2/15

529115/529115 [=====] - 227s 429us/sample - loss: 0.5079 - acc: 0.8517

Epoch 3/15

529115/529115 [=====] - 211s 398us/sample - loss: 0.4919 - acc: 0.8560

```
Epoch 4/15
529115/529115 [=====] - 216s 408us/sample - loss: 0.4838 - acc: 0.8582
Epoch 5/15
529115/529115 [=====] - 226s 426us/sample - loss: 0.4795 - acc: 0.8593
Epoch 6/15
529115/529115 [=====] - 225s 425us/sample - loss: 0.4743 - acc: 0.8611
Epoch 7/15
529115/529115 [=====] - 268s 506us/sample - loss: 0.4696 - acc: 0.8624
Epoch 8/15
529115/529115 [=====] - 246s 464us/sample - loss: 0.4654 - acc: 0.8628
Epoch 9/15
529115/529115 [=====] - 221s 418us/sample - loss: 0.4641 - acc: 0.8621
Epoch 10/15
529115/529115 [=====] - 249s 471us/sample - loss: 0.4596 - acc: 0.8641
Epoch 11/15
529115/529115 [=====] - 254s 480us/sample - loss: 0.4609 - acc: 0.8635
Epoch 12/15
529115/529115 [=====] - 243s 459us/sample - loss: 0.4555 - acc: 0.8657
Epoch 13/15
529115/529115 [=====] - 238s 449us/sample - loss: 0.4509 - acc: 0.8673
Epoch 14/15
529115/529115 [=====] - 281s 531us/sample - loss: 0.4529 - acc: 0.8657
Epoch 15/15
529115/529115 [=====] - 250s 473us/sample - loss: 0.4518 - acc: 0.8658
```

Out[14]: <tensorflow.python.keras.callbacks.History at 0x652009e10>

```
In [0]: 1 testDropoutLoss, testDropoutAcc = model.evaluate(testX, testY, verbose=2)
        2 print('Точность на проверочных данных с learning rate ОПТИМИЗАЦИЕЙ:', testDropoutAcc)
```

```
18724/18724 - 4s - loss: 0.2816 - acc: 0.9214
Точность на проверочных данных с learning rate ОПТИМИЗАЦИЕЙ: 0.92138433
```

Лабораторная работа №3. Реализация сверточной нейронной сети

Данные: В работе предлагается использовать набор данных notMNIST, который состоит из изображений размерностью 28×28 первых 10 букв латинского алфавита (A ... J, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке: https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz (https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz) (большой набор данных); https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz (https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz) (маленький набор данных); Описание данных на английском языке доступно по ссылке: <http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html> (<http://yaroslavvb.blogspot.sg/2011/09/notmnist-dataset.html>)

In [0]:

```
1 import tensorflow as tf
2 from scipy.io import loadmat
3 import numpy as np
4 from keras.utils import to_categorical
5 from tensorflow.keras import layers, models, Sequential
```

Задание 1. Реализуйте нейронную сеть с двумя сверточными слоями, и одним полносвязным с нейронами с кусочно-линейной функцией активации. Какова точность построенной модели?

```
In [0]: 1 def getDataset(file):
        2     matfile = loadmat(file)
        3     x = matfile['images'] / 255
        4     y = matfile['labels']
        5     x, y = x.T, y.T
        6     x = x.reshape(x.shape[0], x.shape[1], x.shape[2], 1)
        7     y = to_categorical(y)
        8     return x, y
```

```
In [0]: 1 trainX, trainY = getDataset("/content/drive/My Drive/Collab Data/train_dataset.mat")
        2 print(trainX.shape, trainY.shape)
        3 testX, testY = getDataset("/content/drive/My Drive/Collab Data/test_dataset.mat")
        4 print(testX.shape, testY.shape)
```

```
(529115, 28, 28, 1) (529115, 1)
(18724, 28, 28, 1) (18724, 1)
```

```
In [0]: 1 model = Sequential([
        2     layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)),
        3     layers.Conv2D(32, kernel_size=3, activation='relu'),
        4     layers.Flatten(),
        5     layers.Dense(10, activation='softmax')
        6 ])
        7
        8 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        9
       10 model.fit(trainX, trainY, validation_data=(testX, testY), epochs=15)
```

```
Train on 529115 samples, validate on 18724 samples
Epoch 1/15
```



```
529115/529115 [=====] - 52s 98us/sample - loss: 0.3813 - acc: 0.8950 - val_loss:
0.1521 - val_acc: 0.9593
Epoch 2/15
529115/529115 [=====] - 52s 99us/sample - loss: 0.3055 - acc: 0.9151 - val_loss:
0.1340 - val_acc: 0.9636
Epoch 3/15
529115/529115 [=====] - 52s 99us/sample - loss: 0.2730 - acc: 0.9237 - val_loss:
0.1514 - val_acc: 0.9601
Epoch 4/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.2502 - acc: 0.9300 - val_loss:
0.1371 - val_acc: 0.9629
Epoch 5/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.2318 - acc: 0.9345 - val_loss:
0.1356 - val_acc: 0.9632
Epoch 6/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.2173 - acc: 0.9382 - val_loss:
0.1403 - val_acc: 0.9622
Epoch 7/15
529115/529115 [=====] - 52s 99us/sample - loss: 0.2056 - acc: 0.9412 - val_loss:
0.1465 - val_acc: 0.9623
Epoch 8/15
529115/529115 [=====] - 52s 99us/sample - loss: 0.1951 - acc: 0.9440 - val_loss:
0.1500 - val_acc: 0.9614
Epoch 9/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.1859 - acc: 0.9460 - val_loss:
0.1556 - val_acc: 0.9598
Epoch 10/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.1784 - acc: 0.9480 - val_loss:
0.1649 - val_acc: 0.9589
Epoch 11/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.1716 - acc: 0.9501 - val_loss:
0.1748 - val_acc: 0.9579
Epoch 12/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.1652 - acc: 0.9516 - val_loss:
0.1793 - val_acc: 0.9568
```

```
Epoch 13/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.1597 - acc: 0.9530 - val_loss:
0.1783 - val_acc: 0.9592
Epoch 14/15
529115/529115 [=====] - 52s 99us/sample - loss: 0.1548 - acc: 0.9545 - val_loss:
0.1904 - val_acc: 0.9559
Epoch 15/15
529115/529115 [=====] - 52s 98us/sample - loss: 0.1498 - acc: 0.9558 - val_loss:
0.1948 - val_acc: 0.9586
```

Out[14]: <tensorflow.python.keras.callbacks.History at 0x7fddbaf5c0>

```
In [0]: 1 testLoss, testAcc = model.evaluate(testX, testY, verbose=2)
        2 print('Точность на проверочных данных:', testAcc)
```

```
18724/18724 - 1s - loss: 0.1948 - acc: 0.9586
Точность на проверочных данных: 0.9586093
```

Задание 2. Замените один из сверточных слоев на слой, реализующий операцию пулинга (Pooling) с функцией максимума или среднего. Как это повлияло на точность классификатора?

```
In [0]: 1 modelWithPooling = Sequential([
        2     layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)),
        3     layers.MaxPooling2D((2, 2)),
        4     layers.Flatten(),
        5     layers.Dense(10, activation='softmax')
        6 ])
        7
        8 modelWithPooling.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        9
       10 modelWithPooling.fit(trainX, trainY, validation_data=(testX, testY), epochs=15)
```

Train on 529115 samples, validate on 18724 samples

```
train on 529115 samples, validate on 16724 samples
```

```
Epoch 1/15
```

```
529115/529115 [=====] - 50s 94us/sample - loss: 0.4351 - acc: 0.8830 - val_loss: 0.1937 - val_acc: 0.9501
```

```
Epoch 2/15
```

```
529115/529115 [=====] - 47s 89us/sample - loss: 0.3723 - acc: 0.8998 - val_loss: 0.1731 - val_acc: 0.9541
```

```
Epoch 3/15
```

```
529115/529115 [=====] - 48s 90us/sample - loss: 0.3494 - acc: 0.9053 - val_loss: 0.1665 - val_acc: 0.9560
```

```
Epoch 4/15
```

```
529115/529115 [=====] - 48s 90us/sample - loss: 0.3350 - acc: 0.9088 - val_loss: 0.1646 - val_acc: 0.9566
```

```
Epoch 5/15
```

```
529115/529115 [=====] - 47s 89us/sample - loss: 0.3243 - acc: 0.9116 - val_loss: 0.1615 - val_acc: 0.9584
```

```
Epoch 6/15
```

```
529115/529115 [=====] - 47s 90us/sample - loss: 0.3168 - acc: 0.9132 - val_loss: 0.1600 - val_acc: 0.9581
```

```
Epoch 7/15
```

```
529115/529115 [=====] - 47s 89us/sample - loss: 0.3101 - acc: 0.9149 - val_loss: 0.1565 - val_acc: 0.9586
```

```
Epoch 8/15
```

```
529115/529115 [=====] - 47s 88us/sample - loss: 0.3053 - acc: 0.9159 - val_loss: 0.1600 - val_acc: 0.9576
```

```
Epoch 9/15
```

```
529115/529115 [=====] - 47s 89us/sample - loss: 0.3009 - acc: 0.9171 - val_loss: 0.1610 - val_acc: 0.9579
```

```
Epoch 10/15
```

```
529115/529115 [=====] - 48s 91us/sample - loss: 0.2972 - acc: 0.9181 - val_loss: 0.1594 - val_acc: 0.9568
```

```
Epoch 11/15
```

```
529115/529115 [=====] - 47s 90us/sample - loss: 0.2938 - acc: 0.9187 - val_loss: 0.1602 - val_acc: 0.9564
```

```
Epoch 12/15
```

```
529115/529115 [=====] - 47s 89us/sample - loss: 0.2913 - acc: 0.9191 - val_loss:
```

```

0.1607 - val_acc: 0.9573
Epoch 13/15
529115/529115 [=====] - 47s 89us/sample - loss: 0.2890 - acc: 0.9199 - val_loss:
0.1639 - val_acc: 0.9557
Epoch 14/15
529115/529115 [=====] - 48s 90us/sample - loss: 0.2867 - acc: 0.9205 - val_loss:
0.1625 - val_acc: 0.9575
Epoch 15/15
529115/529115 [=====] - 47s 89us/sample - loss: 0.2848 - acc: 0.9208 - val_loss:
0.1619 - val_acc: 0.9567

```

Out[17]: <tensorflow.python.keras.callbacks.History at 0x7fddb3eeeb38>

```

In [0]: 1 testWithPoolingLoss, testWithPoolingAcc = modelWithPooling.evaluate(testX, testY, verbose=2)
        2 print('Точность на проверочных данных с пулингом:', testWithPoolingAcc)

```

```

18724/18724 - 1s - loss: 0.1619 - acc: 0.9567
Точность на проверочных данных с пулингом: 0.95674

```

Задание 3. Реализуйте классическую архитектуру сверточных сетей LeNet-5 (<http://yann.lecun.com/exdb/lenet/>).

```

In [0]: 1 leNet5model = Sequential([
        2     layers.Conv2D(filters=6, kernel_size=3, activation='relu', input_shape=(28,28,1)),
        3     layers.AveragePooling2D(),
        4     layers.Conv2D(filters=16, kernel_size=3, activation='relu'),
        5     layers.AveragePooling2D(),
        6     layers.Flatten(),
        7     layers.Dense(units=120, activation='relu'),
        8     layers.Dense(units=84, activation='relu'),
        9     layers.Dense(units=10, activation = 'softmax')
       10 ])
       11
       12 leNet5model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])

```

```
13  
14 leNet5model.fit(trainX, trainY, validation_data=(testX, testY), epochs=15)
```

Train on 529115 samples, validate on 18724 samples

Epoch 1/15

529115/529115 [=====] - 49s 93us/sample - loss: 0.4170 - acc: 0.8726 - val_loss: 0.1603 - val_acc: 0.9530

Epoch 2/15

529115/529115 [=====] - 48s 91us/sample - loss: 0.3040 - acc: 0.9059 - val_loss: 0.1341 - val_acc: 0.9592

Epoch 3/15

529115/529115 [=====] - 48s 91us/sample - loss: 0.2758 - acc: 0.9140 - val_loss: 0.1201 - val_acc: 0.9630

Epoch 4/15

529115/529115 [=====] - 48s 90us/sample - loss: 0.2587 - acc: 0.9191 - val_loss: 0.1198 - val_acc: 0.9651

Epoch 5/15

529115/529115 [=====] - 48s 90us/sample - loss: 0.2476 - acc: 0.9223 - val_loss: 0.1181 - val_acc: 0.9647

Epoch 6/15

529115/529115 [=====] - 48s 91us/sample - loss: 0.2382 - acc: 0.9249 - val_loss: 0.1159 - val_acc: 0.9656

Epoch 7/15

529115/529115 [=====] - 49s 92us/sample - loss: 0.2315 - acc: 0.9267 - val_loss: 0.1096 - val_acc: 0.9677

Epoch 8/15

529115/529115 [=====] - 48s 91us/sample - loss: 0.2251 - acc: 0.9288 - val_loss: 0.1109 - val_acc: 0.9672

Epoch 9/15

529115/529115 [=====] - 48s 90us/sample - loss: 0.2200 - acc: 0.9301 - val_loss: 0.1083 - val_acc: 0.9682

Epoch 10/15

529115/529115 [=====] - 48s 90us/sample - loss: 0.2158 - acc: 0.9314 - val_loss:

```
0.1093 - val_acc: 0.9678
Epoch 11/15
529115/529115 [=====] - 48s 90us/sample - loss: 0.2113 - acc: 0.9327 - val_loss:
0.1108 - val_acc: 0.9688
Epoch 12/15
529115/529115 [=====] - 48s 90us/sample - loss: 0.2071 - acc: 0.9338 - val_loss:
0.1110 - val_acc: 0.9675
Epoch 13/15
529115/529115 [=====] - 48s 90us/sample - loss: 0.2041 - acc: 0.9348 - val_loss:
0.1129 - val_acc: 0.9674
Epoch 14/15
529115/529115 [=====] - 49s 92us/sample - loss: 0.2010 - acc: 0.9355 - val_loss:
0.1109 - val_acc: 0.9682
Epoch 15/15
529115/529115 [=====] - 48s 90us/sample - loss: 0.1984 - acc: 0.9363 - val_loss:
0.1137 - val_acc: 0.9668
```

Out[19]: <tensorflow.python.keras.callbacks.History at 0x7fddb3f58860>

```
In [0]: 1 testLeNetLoss, testLeNetAcc = leNet5model.evaluate(testX, testY, verbose=2)
        2 print('Точность на проверочных данных с пулингом:', testLeNetAcc)
```

```
18724/18724 - 1s - loss: 0.1137 - acc: 0.9668
Точность на проверочных данных с пулингом: 0.966834
```

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> (<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/>)

```
In [0]: 1 from tensorflow.keras.datasets import mnist
        2 import matplotlib.pyplot as plt
        3 from tensorflow.keras.models import Sequential
        4 import tensorflow.keras.layers
        5 from tensorflow.keras.utils import to_categorical
        6 from scipy.io import loadmat
        7 import tensorflow as tf
        8 import numpy as np
        9 import matplotlib.pyplot as plt
        10 import json
        11 from yolo.frontend import create_yolo
        12 from yolo.backend.utils.box import draw_scaled_boxes
        13 import os
        14 import yolo
        15 import cv2
        16
        17 print(tf.version.VERSION)
```

1.14.0

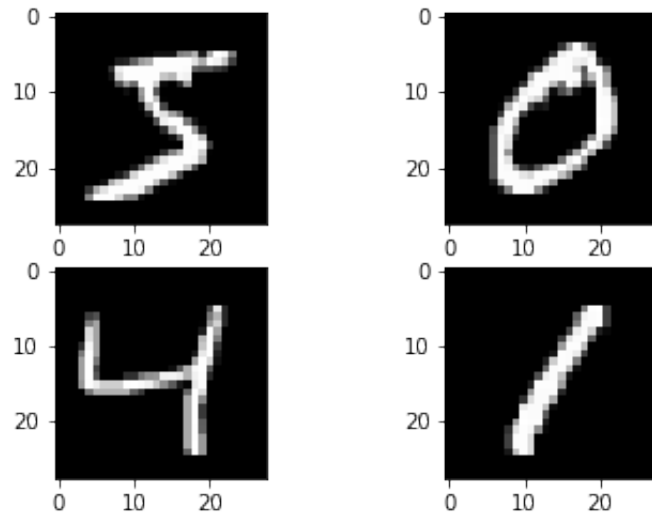
Задание 1. Реализуйте глубокую нейронную сеть (полносвязную или сверточную) и обучите ее на синтетических данных (например, наборы MNIST (<http://yann.lecun.com/exdb/mnist/>) или notMNIST). Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (<http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf>) (<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)).

In [0]:

```
1 def showImages(trainDataset):  
2     plt.subplot(221)  
3     plt.imshow(trainDataset[0], cmap=plt.get_cmap('gray'))  
4     plt.subplot(222)  
5     plt.imshow(trainDataset[1], cmap=plt.get_cmap('gray'))  
6     plt.subplot(223)  
7     plt.imshow(trainDataset[2], cmap=plt.get_cmap('gray'))  
8     plt.subplot(224)  
9     plt.imshow(trainDataset[3], cmap=plt.get_cmap('gray'))  
10    plt.show()
```

```
In [0]: 1 (xTrain, yTrain), (xTest, yTest) = mnist.load_data()  
        2 showImages(xTrain)
```

Downloading data from <https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>
(<https://storage.googleapis.com/tensorflow/tf-keras-datasets/mnist.npz>)
11493376/11490434 [=====] - 0s 0us/step



```
In [0]: 1 print(xTrain.shape, yTrain.shape)  
        2 print(xTest.shape, yTest.shape)
```

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

```
In [0]: 1 def largerModel(num_classes):
2         model = tf.keras.models.Sequential([
3             tf.keras.layers.Conv2D(30, (5, 5), input_shape=(28, 28, 1), activation='relu'),
4             tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
5             tf.keras.layers.Conv2D(15, (3, 3), activation='relu'),
6             tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
7             tf.keras.layers.Dropout(0.2),
8             tf.keras.layers.Flatten(),
9             tf.keras.layers.Dense(128, activation='relu'),
10            tf.keras.layers.Dense(50, activation='relu'),
11            tf.keras.layers.Dense(num_classes, activation='softmax')
12        ])
13
14        model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
15        return model
16
17 def reshapeDataset(x, y):
18     x = x / 255
19     x = x.reshape(x.shape[0], x.shape[1], x.shape[2], 1)
20     y = to_categorical(y)
21     return x, y
```

```
In [0]: 1 (xTrain, yTrain), (xTest, yTest) = mnist.load_data()
2     xTrain, yTrain = reshapeDataset(xTrain, yTrain)
3     xTest, yTest = reshapeDataset(xTest, yTest)
4
5     def trainModel(xTrain, yTrain, xTest, yTest):
6         num_classes = yTrain.shape[1]
7         model = largerModel(num_classes)
8
9         model.fit(xTrain, yTrain, epochs=10, validation_data=(xTest, yTest))
```

```

10
11 scores = model.evaluate(xTest, yTest, verbose=0)
12 print("Large CNN Error: %.2f%%" % (100-scores[1]*100))
13
14 trainModel(xTrain, yTrain, xTest, yTest)

```

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/init_ops.py:1251: calling VarianceScaling.__init__ (from tensorflow.python.ops.init_ops) with dtype is deprecated and will be removed in a future version.

Instructions for updating:

Call initializer instance with the dtype argument instead of passing it to the constructor

Train on 60000 samples, validate on 10000 samples

Epoch 1/10

60000/60000 [=====] - 28s 460us/sample - loss: 0.1946 - acc: 0.9394 - val_loss: 0.0537 - val_acc: 0.9814

Epoch 2/10

60000/60000 [=====] - 27s 455us/sample - loss: 0.0666 - acc: 0.9790 - val_loss: 0.0346 - val_acc: 0.9889

Epoch 3/10

60000/60000 [=====] - 27s 457us/sample - loss: 0.0513 - acc: 0.9840 - val_loss: 0.0344 - val_acc: 0.9885

Epoch 4/10

60000/60000 [=====] - 29s 479us/sample - loss: 0.0430 - acc: 0.9867 - val_loss: 0.0321 - val_acc: 0.9891

Epoch 5/10

60000/60000 [=====] - 30s 498us/sample - loss: 0.0349 - acc: 0.9887 - val_loss: 0.0389 - val_acc: 0.9886

Epoch 6/10

60000/60000 [=====] - 30s 507us/sample - loss: 0.0325 - acc: 0.9894 - val_loss: 0.0260 - val_acc: 0.9915

Epoch 7/10

60000/60000 [=====] - 28s 465us/sample - loss: 0.0290 - acc: 0.9905 - val_loss: 0.0239 - val_acc: 0.9923

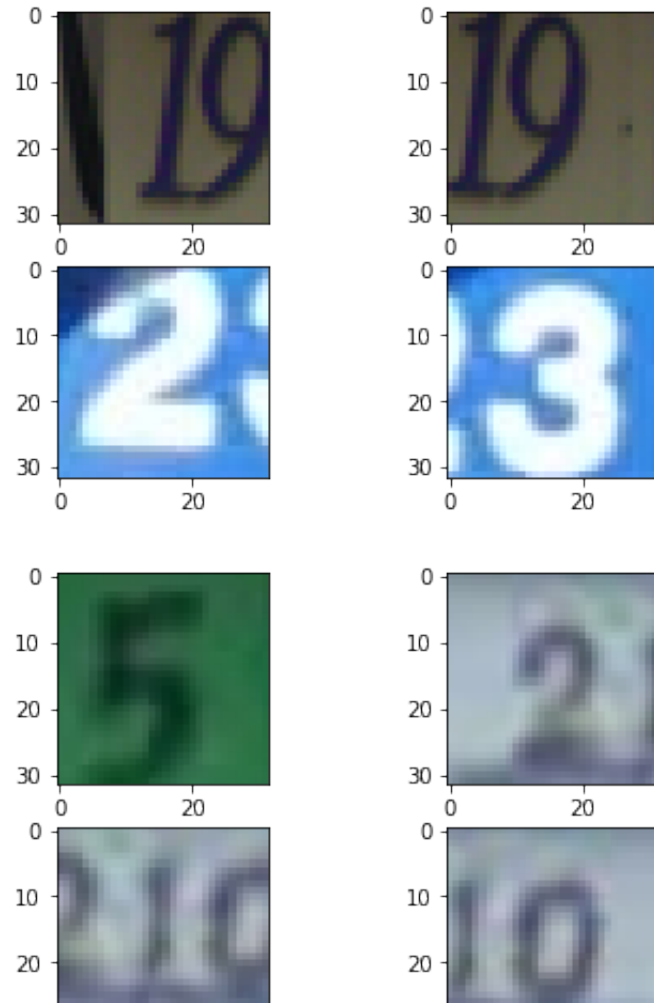
```
Epoch 8/10
60000/60000 [=====] - 28s 463us/sample - loss: 0.0260 - acc: 0.9916 - val_loss:
0.0274 - val_acc: 0.9911
Epoch 9/10
60000/60000 [=====] - 28s 463us/sample - loss: 0.0243 - acc: 0.9923 - val_loss:
0.0245 - val_acc: 0.9917
Epoch 10/10
60000/60000 [=====] - 28s 466us/sample - loss: 0.0218 - acc: 0.9925 - val_loss:
0.0293 - val_acc: 0.9912
Large CNN Error: 0.88%
```

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

```
In [0]: 1 def getDataset(file):
2         matplotlib = loadmat(file)
3         x = matplotlib['X'] / 255
4         y = matplotlib['y']
5         y = to_categorical(y)
6         return x, y
7
8 xTrain, yTrain = getDataset("/content/drive/My Drive/Collab Data/train_32x32.mat")
9 xTest, yTest = getDataset("/content/drive/My Drive/Collab Data/test_32x32.mat")
```

```
In [0]: 1 def showGoogleImages(trainDataset):
2         plt.subplot(221)
3         plt.imshow(trainDataset[:, :, :, 0], cmap=plt.get_cmap('gray'))
4         plt.subplot(222)
5         plt.imshow(trainDataset[:, :, :, 1], cmap=plt.get_cmap('gray'))
6         plt.subplot(223)
7         plt.imshow(trainDataset[:, :, :, 2], cmap=plt.get_cmap('gray'))
8         plt.subplot(224)
```

```
9 plt.imshow(trainDataset[:, :, :, 3], cmap=plt.get_cmap('gray'))
10 plt.show()
11
12 showGoogleImages(xTrain)
13 showGoogleImages(xTest)
```





```
In [0]: 1 xTrain, xTest = np.moveaxis(xTrain, -1, 0), np.moveaxis(xTest, -1, 0)
        2 yTrain, yTest = np.where(yTrain==10, 0, yTrain), np.where(yTest==10, 0, yTest)
        3 print(xTrain.shape, yTrain.shape)
        4 print(xTest.shape, yTest.shape)
```

```
(73257, 32, 32, 3) (73257, 11)
(26032, 32, 32, 3) (26032, 11)
```

```
In [0]: 1 def largerGoogleModel(numClasses):
        2     model = tf.keras.models.Sequential([
        3         tf.keras.layers.Conv2D(30, (5, 5), input_shape=(32, 32, 3), activation='relu'),
        4         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        5         tf.keras.layers.Conv2D(15, (3, 3), activation='relu'),
        6         tf.keras.layers.MaxPooling2D(pool_size=(2, 2)),
        7         tf.keras.layers.Dropout(0.2),
        8         tf.keras.layers.Flatten(),
        9         tf.keras.layers.Dense(128, activation='relu'),
        10        tf.keras.layers.Dense(50, activation='relu'),
        11        tf.keras.layers.Dense(11, activation='softmax')
        12    ])
        13
        14
        15    model.compile(loss='categorical_crossentropy', optimizer='adam', metrics=['accuracy'])
        16    return model
        17
        18
        19 def trainGoogleModel(xTrain, yTrain, xTest, yTest):
        20     print(xTrain.shape, yTrain.shape, xTest.shape, yTest.shape)
        21     numClasses = yTrain.shape[1]
        22     googleModel = largerGoogleModel(numClasses)
        23     googleModel.fit(xTrain, yTrain, validation_data=(xTest, yTest), epochs=15, batch_size=200)
```

```
24
25     scores = googleModel.evaluate(xTest, yTest, verbose=0)
26     print("Large CNN Error: %.2f%%" % (100-scores[1]*100))
27     return googleModel
28
29 googleModel = trainGoogleModel(xTrain, yTrain, xTest, yTest)
```

(73257, 32, 32, 3) (73257, 11) (26032, 32, 32, 3) (26032, 11)

Train on 73257 samples, validate on 26032 samples

Epoch 1/15

73257/73257 [=====] - 48s 661us/sample - loss: 1.5910 - acc: 0.4589 - val_loss: 0.7689 - val_acc: 0.7797

Epoch 2/15

73257/73257 [=====] - 46s 632us/sample - loss: 0.7406 - acc: 0.7814 - val_loss: 0.6227 - val_acc: 0.8294

Epoch 3/15

73257/73257 [=====] - 46s 629us/sample - loss: 0.6164 - acc: 0.8181 - val_loss: 0.5644 - val_acc: 0.8384

Epoch 4/15

73257/73257 [=====] - 47s 635us/sample - loss: 0.5584 - acc: 0.8338 - val_loss: 0.5286 - val_acc: 0.8525

Epoch 5/15

73257/73257 [=====] - 47s 640us/sample - loss: 0.5180 - acc: 0.8473 - val_loss: 0.4854 - val_acc: 0.8616

Epoch 6/15

73257/73257 [=====] - 48s 653us/sample - loss: 0.4817 - acc: 0.8556 - val_loss: 0.4648 - val_acc: 0.8668

Epoch 7/15

73257/73257 [=====] - 46s 630us/sample - loss: 0.4543 - acc: 0.8611 - val_loss: 0.4515 - val_acc: 0.8715


```
Epoch 8/15
73257/73257 [=====] - 46s 634us/sample - loss: 0.4301 - acc: 0.8714 - val_loss:
0.4609 - val_acc: 0.8711
Epoch 9/15
73257/73257 [=====] - 46s 630us/sample - loss: 0.4089 - acc: 0.8754 - val_loss:
0.4204 - val_acc: 0.8813
Epoch 10/15
73257/73257 [=====] - 47s 642us/sample - loss: 0.3880 - acc: 0.8826 - val_loss:
0.4058 - val_acc: 0.8837
Epoch 11/15
73257/73257 [=====] - 48s 651us/sample - loss: 0.3732 - acc: 0.8859 - val_loss:
0.4054 - val_acc: 0.8852
Epoch 12/15
73257/73257 [=====] - 46s 634us/sample - loss: 0.3593 - acc: 0.8910 - val_loss:
0.4015 - val_acc: 0.8856
Epoch 13/15
73257/73257 [=====] - 47s 636us/sample - loss: 0.3479 - acc: 0.8934 - val_loss:
0.3937 - val_acc: 0.8890
Epoch 14/15
73257/73257 [=====] - 46s 628us/sample - loss: 0.3362 - acc: 0.8969 - val_loss:
0.4011 - val_acc: 0.8839
Epoch 15/15
73257/73257 [=====] - 47s 638us/sample - loss: 0.3299 - acc: 0.8985 - val_loss:
0.3822 - val_acc: 0.8929
Large CNN Error: 10.71%
```

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/>)). Пример использования библиотеки TensorFlow на смартфоне можете воспользоваться демонстрационным приложением от Google (<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/ios> (<https://github.com/tensorflow/tensorflow/tree/master/tensorflow/examples/ios>)).

```
In [0]: 1 # model.save('/content/drive/My Drive/Collab Data/NumbersModel.h5')
```

```
In [0]: 1 ! git clone https://github.com/penny4860/Yolo-digit-detector
```

```
In [0]: 1 os.chdir('/content/Yolo-digit-detector')  
2 !pip install -r requirements.txt
```

```
In [0]: 1 os.chdir('/content/Yolo-digit-detector')  
2 ! pip install -e .
```

```
In [0]: 1 yolo_detector = create_yolo("ResNet50", ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"], 416)
```

Layer (type)	Output Shape	Param #	Connected to
=====			
input_2 (InputLayer)	(None, 416, 416, 3)	0	
conv1 (Conv2D)	(None, 208, 208, 64)	9472	input_2[0][0]
bn_conv1 (BatchNormalization)	(None, 208, 208, 64)	256	conv1[0][0]
activation_50 (Activation)	(None, 208, 208, 64)	0	bn_conv1[0][0]
max_pooling2d_2 (MaxPooling2D)	(None, 103, 103, 64)	0	activation_50[0][0]
res2a_branch2a (Conv2D)	(None, 103, 103, 64)	4160	max_pooling2d_2[0][0]
bn2a_branch2a (BatchNormalization)	(None, 103, 103, 64)	256	res2a_branch2a[0][0]
activation_51 (Activation)	(None, 103, 103, 64)	0	bn2a_branch2a[0][0]
res2a_branch2b (Conv2D)	(None, 103, 103, 64)	3680	activation_51[0][0]

```
In [0]: 1 DEFAULT_WEIGHT_FILE = os.path.join(yolo.PROJECT_ROOT, "weights.h5")
2 yolo_detector.load_weights(DEFAULT_WEIGHT_FILE)
```

Loading pre-trained weights in /content/Yolo-digit-detector/weights.h5

```
In [0]: 1 DEFAULT_IMAGE_FOLDER = os.path.join(yolo.PROJECT_ROOT, "tests", "dataset", "svhn", "imgs")
2
3 img_files = [os.path.join(DEFAULT_IMAGE_FOLDER, "1.png"), os.path.join(DEFAULT_IMAGE_FOLDER, "2.png")]
4 imgs = []
5 for fname in img_files:
6     img = cv2.imread(fname)
7     img = cv2.cvtColor(img, cv2.COLOR_BGR2RGB)
```

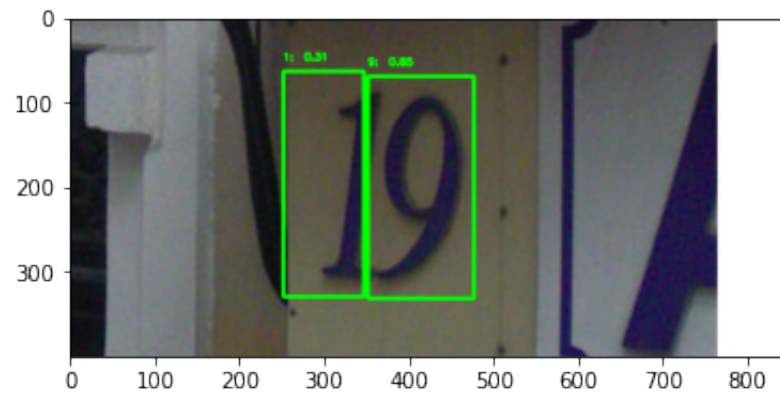
```
8     imgs.append(img)
9     plt.imshow(img)
10    plt.show()
```



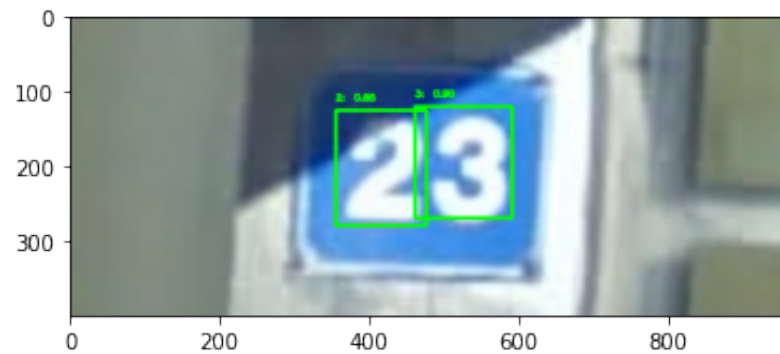
```
In [0]: 1 THRESHOLD = 0.3
        2 for img in imgs:
        3     boxes, probs = yolo_detector.predict(img, THRESHOLD)
        4
        5     # 4. save detection result
        6     image = draw_scaled_boxes(img,
```

```
7         boxes,  
8         probs,  
9         ["0", "1", "2", "3", "4", "5", "6", "7", "8", "9"])  
10  
11     print("{}-boxes are detected.".format(len(boxes)))  
12     plt.imshow(image)  
13     plt.show()
```

2-boxes are detected.

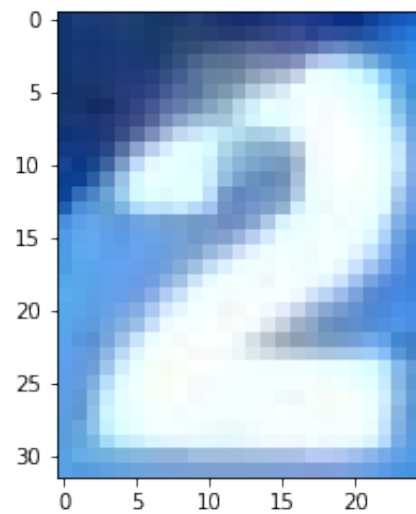


2-boxes are detected.



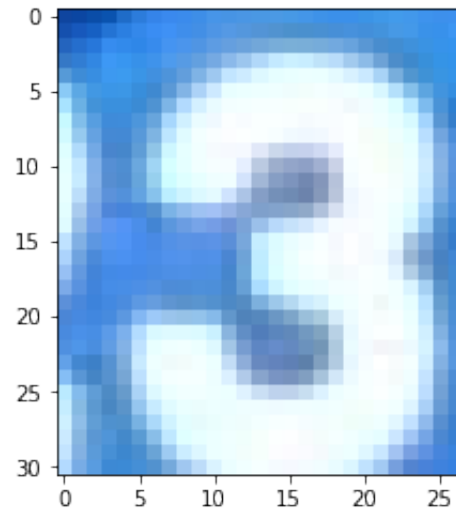
```
In [0]: 1 image = imgs[1]
        2 newImages = []
        3 for box in boxes:
        4     newImages.append(image[box[1]:box[3], box[0]:box[2]])
        5
        6 plt.imshow(newImages[0])
```

Out[66]: <matplotlib.image.AxesImage at 0x7fdb4aecba8>



```
In [0]: 1 plt.imshow(newImages[1])
```

```
Out[67]: <matplotlib.image.AxesImage at 0x7fdcb4e9dbe0>
```



```
In [0]: 1 for image in newImages:
2     res = cv2.resize(image, dsize=(32, 32), interpolation=cv2.INTER_CUBIC)
3     res = res.reshape(1, 32, 32, 3)
4     print(googleModel.predict(res))
```

```
[[0. 0. 1. 0. 0. 0. 0. 0. 0. 0. 0.]]
[[0. 0. 0. 1. 0. 0. 0. 0. 0. 0. 0.]]
```

Задание 4. Реализуйте приложение для ОС iOS, которое может распознавать цифры в номерах домов, используя разработанный ранее классификатор. Какова доля правильных классификаций?


/Приложение для распознавания рукописных цифр/



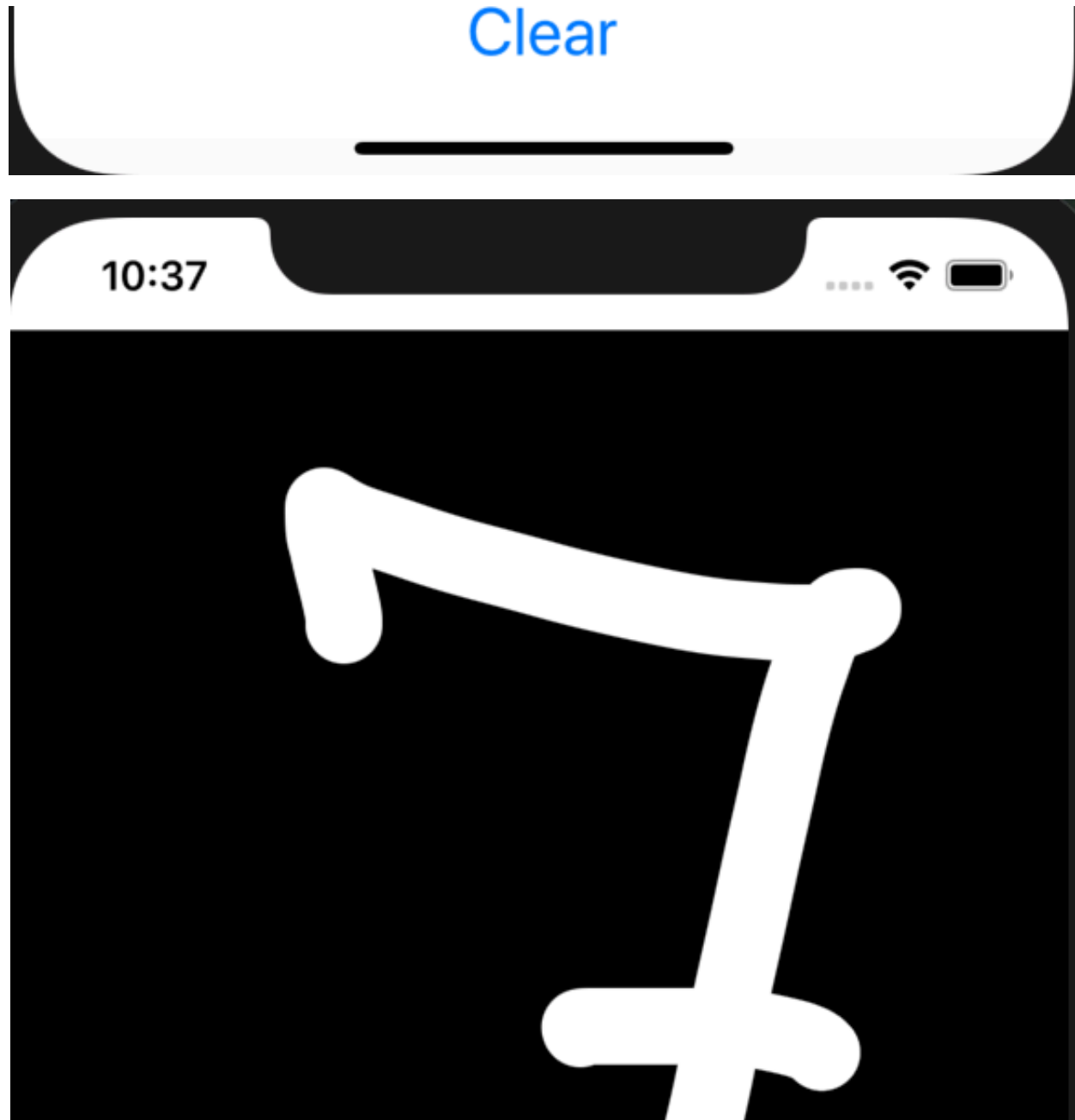
Predicted: 3
Confidence: 0.70687664

Clear



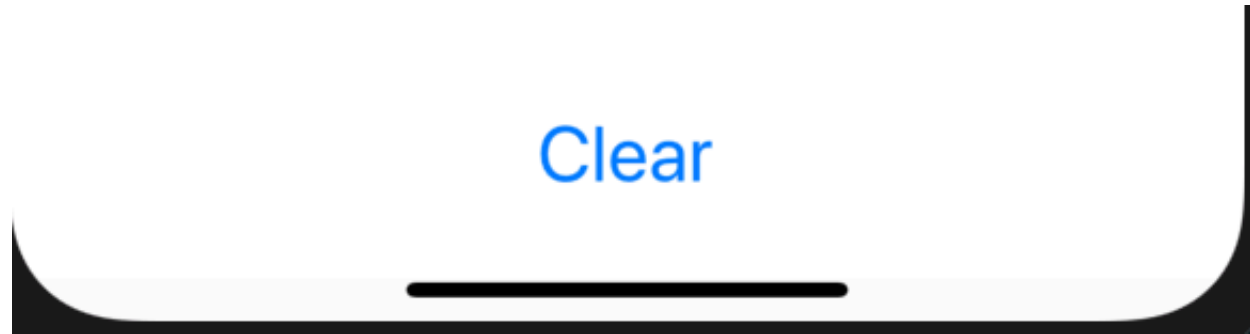


Predicted: 5
Confidence: 0.86782455





Predicted: 7
Confidence: 0.96305645



Лабораторная работа №5. Применение сверточных нейронных сетей (бинарная классификация)

Данные: 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>).

In [0]:

```
1 import os
2 from keras.preprocessing.image import ImageDataGenerator
3 import pandas as pd
4 import PIL
5 import numpy as np
6 from tqdm import tqdm
7 from keras.models import Sequential
8 from keras.layers import Dense, Dropout, Activation, Flatten
9 from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D
10 from keras.models import Model
11 from keras.applications.vgg16 import VGG16
12 from keras.layers import Input
```

Задание 1. Загрузите данные. Разделите исходный набор данных на обучающую, валидационную и контрольную выборки.

In [0]:

```
1 os.chdir('/content/drive/My Drive/kaggle/dogs-vs-cats/train/train')
2 df = pd.DataFrame(columns=['animal', 'image'])
3
4 items = os.listdir()
5 for imageName in tqdm(items):
6     animal = 0 if imageName.split(".")[0] == "cat" else 1
7     df = df.append({'animal': animal, 'image': imageName}, ignore_index=True)
8
9 print(df)
```

```
In [0]: 1 from google.colab import files
        2
        3 df.to_csv('trainPaths.csv')
        4 files.download('trainPaths.csv')
```

```
In [0]: 1 df = pd.read_csv('/content/drive/My Drive/kaggle/dogs-vs-cats/trainPaths.csv')
        2 print(df)
```

```
      Unnamed: 0  animal      image
0              0        0  cat.9578.jpg
1              1        0  cat.9563.jpg
2              2        0  cat.9544.jpg
3              3        0  cat.9574.jpg
4              4        0  cat.9561.jpg
...           ...      ...
24996          24996      1  dog.10151.jpg
24997          24997      1  dog.10121.jpg
24998          24998      1  dog.10117.jpg
24999          24999      1  dog.10147.jpg
25000          25000      1  dog.10138.jpg
```

```
[25001 rows x 3 columns]
```


In [0]:

```
1 train_datagen = ImageDataGenerator(rescale=1./255., validation_split=0.25)
2 test_datagen = ImageDataGenerator(rescale=1./255., validation_split=0.75)
3
4 train_generator = train_datagen.flow_from_dataframe(
5     dataframe=df,
6     directory='/content/drive/My Drive/kaggle/dogs-vs-cats/train/train',
7     x_col="image",
8     y_col="animal",
9     target_size=(150, 150),
10    batch_size=32,
11    class_mode='raw')
12
13 test_generator = test_datagen.flow_from_dataframe(
14     dataframe=df,
15     directory='/content/drive/My Drive/kaggle/dogs-vs-cats/train/train',
16     x_col="image",
17     y_col="animal",
18     target_size=(150, 150),
19     batch_size=32,
20     class_mode='raw')
```

/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/dataframe_iterator.py:273: UserWarning:
Found 3 invalid image filename(s) in x_col="image". These filename(s) will be ignored.

.format(n_invalid, x_col)

Found 24998 validated image filenames.

Found 25000 validated image filenames.

/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/dataframe_iterator.py:273: UserWarning:
Found 1 invalid image filename(s) in x_col="image". These filename(s) will be ignored.

.format(n_invalid, x_col)

Задание 2. Реализуйте глубокую нейронную сеть с как минимум тремя сверточными слоями. Какое качество классификации получено?

```
In [0]: 1 model = Sequential()
2 model.add(Conv2D(32, (5, 5), activation='relu', padding='same', input_shape=(150, 150, 3)))
3 model.add(MaxPooling2D((2, 2)))
4 model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
5 model.add(MaxPooling2D((2, 2)))
6 model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
7 model.add(MaxPooling2D((2, 2)))
8 model.add(Flatten())
9 model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
10 model.add(Dense(1, activation='sigmoid'))
11
12 model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
13
14 model.fit_generator(
15     train_generator,
16     steps_per_epoch=20,
17     epochs=5,
18     validation_data=test_generator,
19     validation_steps=8)
```

Epoch 1/5

20/20 [=====] - 435s 22s/step - loss: 0.9911 - acc: 0.4875 - val_loss: 0.6926 - val_acc: 0.5078

Epoch 2/5

4/20 [=====>.....] - ETA: 6:02 - loss: 0.6943 - acc: 0.4297

/usr/local/lib/python3.6/dist-packages/keras/utils/data_utils.py:610: UserWarning: The input 403 could not be retrieved. It could be because a worker has died.

UserWarning)

```
20/20 [=====] - 443s 22s/step - loss: 0.6942 - acc: 0.4766 - val_loss: 0.6925 -  
val_acc: 0.5625  
Epoch 3/5  
20/20 [=====] - 378s 19s/step - loss: 0.6943 - acc: 0.5000 - val_loss: 0.6934 -  
val_acc: 0.4805  
Epoch 4/5  
20/20 [=====] - 434s 22s/step - loss: 0.6931 - acc: 0.5250 - val_loss: 0.7036 -  
val_acc: 0.4883  
Epoch 5/5  
20/20 [=====] - 424s 21s/step - loss: 0.6938 - acc: 0.5156 - val_loss: 0.6937 -  
val_acc: 0.4727
```

Out[28]: <keras.callbacks.History at 0x7ff5ba687fd0>

Задание 3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?

```

In [0]: 1 train_datagen = ImageDataGenerator(rescale=1.0/255.0, width_shift_range=0.1, height_shift_range=0.1, ho
2 test_datagen = ImageDataGenerator(rescale=1.0/255.0)
3
4 train_generator = train_datagen.flow_from_dataframe(
5     dataframe=df,
6     directory='/content/drive/My Drive/kaggle/dogs-vs-cats/train/train',
7     x_col="image",
8     y_col="animal",
9     target_size=(150, 150),
10    batch_size=32,
11    class_mode='raw')
12
13 test_generator = test_datagen.flow_from_dataframe(
14     dataframe=df,
15     directory='/content/drive/My Drive/kaggle/dogs-vs-cats/train/train',
16     x_col="image",
17     y_col="animal",
18     target_size=(150, 150),
19     batch_size=32,
20     class_mode='raw')

```

```

/usr/local/lib/python3.6/dist-packages/keras_preprocessing/image/dataframe_iterator.py:273: UserWarning:
Found 1 invalid image filename(s) in x_col="image". These filename(s) will be ignored.
    .format(n_invalid, x_col)

```

Found 25000 validated image filenames.

Found 25000 validated image filenames.

```

In [0]: 1 model = Sequential()
2 model.add(Conv2D(32, (5, 5), activation='relu', padding='same', input_shape=(150, 150, 3)))
3 model.add(MaxPooling2D((2, 2)))
4 model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))

```

```

4 model.add(Conv2D(64, (3, 3), activation='relu', padding='same'))
5 model.add(MaxPooling2D((2, 2)))
6 model.add(Conv2D(128, (3, 3), activation='relu', padding='same'))
7 model.add(MaxPooling2D((2, 2)))
8 model.add(Flatten())
9 model.add(Dense(128, activation='relu', kernel_initializer='he_uniform'))
10 model.add(Dense(1, activation='sigmoid'))
11
12 model.compile(optimizer="adam", loss='binary_crossentropy', metrics=['accuracy'])
13
14 model.fit_generator(
15     train_generator,
16     steps_per_epoch=20,
17     epochs=5,
18     validation_data=test_generator,
19     validation_steps=8)

```

Epoch 1/5

20/20 [=====] - 365s 18s/step - loss: 0.7956 - acc: 0.4953 - val_loss: 0.6920 - val_acc: 0.5703

Epoch 2/5

20/20 [=====] - 350s 17s/step - loss: 0.6935 - acc: 0.5016 - val_loss: 0.6904 - val_acc: 0.5195

Epoch 3/5

17/20 [=====>.....] - ETA: 52s - loss: 0.6938 - acc: 0.4816

/usr/local/lib/python3.6/dist-packages/keras/utils/data_utils.py:610: UserWarning: The input 772 could not be retrieved. It could be because a worker has died.

UserWarning)

20/20 [=====] - 408s 20s/step - loss: 0.6935 - acc: 0.4906 - val_loss: 0.6934 - val_acc: 0.4805

Epoch 4/5

```
20/20 [=====] - 349s 17s/step - loss: 0.6920 - acc: 0.5203 - val_loss: 0.6898 -  
val_acc: 0.5078  
Epoch 5/5  
20/20 [=====] - 347s 17s/step - loss: 0.6919 - acc: 0.5422 - val_loss: 0.6885 -  
val_acc: 0.5430
```

```
Out[30]: <keras.callbacks.History at 0x7ff5b9c93dd8>
```

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?

```
In [0]: 1 from keras.applications import MobileNet
        2
        3 base_model=MobileNet(weights='imagenet',include_top=False)
        4
        5 x=base_model.output
        6 x=GlobalAveragePooling2D()(x)
        7 x=Dense(1024,activation='relu')(x)
        8 x=Dense(1024,activation='relu')(x)
        9 x=Dense(512,activation='relu')(x)
       10 preds=Dense(1,activation='softmax')(x)
       11 model=Model(inputs=base_model.input,outputs=preds)
       12 model.summary()
```

/usr/local/lib/python3.6/dist-packages/keras_applications/mobilenet.py:207: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.

warnings.warn("`input_shape` is undefined or non-square, '

Model: "model_3"

Layer (type)	Output Shape	Param #
=====		
input_5 (InputLayer)	(None, None, None, 3)	0

conv1_pad (ZeroPadding2D)	(None, None, None, 3)	0

conv1 (Conv2D)	(None, None, None, 32)	864

conv1_bn (BatchNormalization)	(None, None, None, 32)	128

conv1_relu (ReLU)	(None, None, None, 32)	0

conv_dw_1 (DepthwiseConv2D)	(None, None, None, 32)	288

```
In [0]: 1 for layer in model.layers:
        2     layer.trainable=False
```

```
In [0]: 1 model.compile(optimizer='Adam',loss='binary_crossentropy',metrics=['accuracy'])
        2
        3 model.fit_generator(
        4     train_generator,
        5     steps_per_epoch=20,
        6     epochs=5,
        7     validation_data=test_generator,
        8     validation_steps=8)
```

Epoch 1/5

20/20 [=====] - 273s 14s/step - loss: 8.0459 - acc: 0.4953 - val_loss: 8.3448 - val_acc: 0.4766

Epoch 2/5

20/20 [=====] - 249s 12s/step - loss: 7.8466 - acc: 0.5078 - val_loss: 7.9712 - val_acc: 0.5000

Epoch 3/5

20/20 [=====] - 233s 12s/step - loss: 7.7470 - acc: 0.5141 - val_loss: 8.0335 - val_acc: 0.4961

Epoch 4/5

20/20 [=====] - 214s 11s/step - loss: 7.6723 - acc: 0.5188 - val_loss: 7.5353 - val_acc: 0.5273

Epoch 5/5

20/20 [=====] - 253s 13s/step - loss: 7.9463 - acc: 0.5016 - val_loss: 8.0335 - val_acc: 0.4961

Out[18]: <keras.callbacks.History at 0x7f298a378048>

Лабораторная работа №6. Применение сверточных нейронных сетей (многоклассовая классификация)

Данные: Набор данных для распознавания языка жестов, который состоит из изображений размерности 28x28 в оттенках серого (значение пикселя от 0 до 255). Каждое из изображений обозначает букву латинского алфавита, обозначенную с помощью жеста, как показано на рисунке ниже (рисунок цветной, а изображения в наборе данных в оттенках серого). Обучающая выборка включает в себя 27,455 изображений, а контрольная выборка содержит 7172 изображения. Данные в виде csv-файлов можно скачать на сайте Kaggle -> <https://www.kaggle.com/datamunge/sign-language-mnist> (<https://www.kaggle.com/datamunge/sign-language-mnist>)

```
In [0]: 1 import pandas as pd
        2 import tensorflow as tf
        3 from keras.utils import np_utils
        4 from sklearn.model_selection import train_test_split
        5 from tensorflow.keras import layers, models, Sequential
        6 from sklearn.metrics import roc_auc_score
        7 from keras.layers import Conv2D, MaxPooling2D, GlobalAveragePooling2D, Dense
        8 from keras.preprocessing.image import ImageDataGenerator
        9 from keras.models import Model
       10 from tensorflow.keras.layers import InputLayer
       11 from keras.layers import Concatenate
```

Задание 1. Загрузите данные. Разделите исходный набор данных на обучающую и валидационную выборки.

```
In [0]: 1 df = pd.read_csv('/content/drive/My Drive/kaggle/sign_mnist_train.csv')
        2 print(df)
        3
        4 testDF = pd.read_csv('/content/drive/My Drive/kaggle/sign_mnist_test.csv')
        5 print(testDF)
```

	label	pixel1	pixel2	pixel3	...	pixel781	pixel782	pixel783	pixel784
0	3	107	118	127	...	206	204	203	202
1	6	155	157	156	...	175	103	135	149
2	2	187	188	188	...	198	195	194	195
3	2	211	211	212	...	225	222	229	163
4	13	164	167	170	...	157	163	164	179
...
27450	13	189	189	190	...	234	200	222	225
27451	23	151	154	157	...	195	195	195	194
27452	18	174	174	174	...	203	202	200	200
27453	17	177	181	184	...	47	64	87	93
27454	23	179	180	180	...	197	205	209	215

[27455 rows x 785 columns]

	label	pixel1	pixel2	pixel3	...	pixel781	pixel782	pixel783	pixel784
0	6	149	149	150	...	106	112	120	107
1	5	126	128	131	...	184	184	182	180
2	10	85	88	92	...	226	225	224	222
3	0	203	205	207	...	230	240	253	255
4	3	188	191	193	...	49	46	46	53
...
7167	1	135	119	108	...	184	176	167	163
7168	12	157	159	161	...	210	210	209	208
7169	2	190	191	190	...	210	211	209	208
7170	4	201	205	208	...	91	67	70	63
7171	2	173	174	173	...	195	195	193	192

[7172 rows x 785 columns]

```
In [0]: 1 def getXandY(df):
2       y = df["label"].values.reshape((df.shape[0], 1))
3       x = df.drop(columns="label").values.reshape((df.shape[0], 28, 28, 1))
4       print(y.shape, x.shape)
5       return x, y
6
7 x, y = getXandY(df)
8 tX, tY = getXandY(testDF)
```

```
(27455, 1) (27455, 28, 28, 1)
(7172, 1) (7172, 28, 28, 1)
```

```
In [0]: 1 trainX, testX, trainY, testY = train_test_split(x, y, test_size=.25)
```

Задание 2. Реализуйте глубокую нейронную сеть со сверточными слоями. Какое качество классификации получено? Какая архитектура сети была использована?

```
In [0]: 1 model = Sequential([
2       layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)),
3       layers.Conv2D(32, kernel_size=5, activation='relu'),
4       layers.MaxPooling2D((2, 2)),
5       layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
6       layers.MaxPooling2D((2, 2)),
7       layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
8       layers.MaxPooling2D((2, 2)),
9       layers.Flatten(),
10      layers.Dense(25, activation='softmax')
11  ])
12
13 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
14
15 model.fit(trainX, trainY, validation_data=(testX, testY), epochs=15)
```

```
15 model.fit(train, train, validation_data=(test, test), epochs=15,  
16
```

Train on 20591 samples, validate on 6864 samples

Epoch 1/15

20591/20591 [=====] - 144s 7ms/sample - loss: 1.0571 - acc: 0.7136 - val_loss: 0.0424 - val_acc: 0.9850

Epoch 2/15

20591/20591 [=====] - 145s 7ms/sample - loss: 0.0364 - acc: 0.9891 - val_loss: 0.0588 - val_acc: 0.9768

Epoch 3/15

20591/20591 [=====] - 146s 7ms/sample - loss: 0.0436 - acc: 0.9884 - val_loss: 0.0113 - val_acc: 0.9969

Epoch 4/15

20591/20591 [=====] - 146s 7ms/sample - loss: 0.0145 - acc: 0.9957 - val_loss: 7.3109e-04 - val_acc: 0.9999

Epoch 5/15

20591/20591 [=====] - 147s 7ms/sample - loss: 8.1862e-05 - acc: 1.0000 - val_loss: 6.7947e-04 - val_acc: 0.9999

Epoch 6/15

20591/20591 [=====] - 145s 7ms/sample - loss: 3.6115e-05 - acc: 1.0000 - val_loss: 6.0458e-04 - val_acc: 0.9999

Epoch 7/15

20591/20591 [=====] - 144s 7ms/sample - loss: 2.1497e-05 - acc: 1.0000 - val_loss: 6.1721e-04 - val_acc: 0.9999

Epoch 8/15

20591/20591 [=====] - 144s 7ms/sample - loss: 1.3678e-05 - acc: 1.0000 - val_loss: 5.6933e-04 - val_acc: 0.9999

Epoch 9/15

20591/20591 [=====] - 142s 7ms/sample - loss: 9.1778e-06 - acc: 1.0000 - val_loss: 5.2476e-04 - val_acc: 0.9999

Epoch 10/15

20591/20591 [=====] - 142s 7ms/sample - loss: 6.2680e-06 - acc: 1.0000 - val_loss

```

s: 5.4921e-04 - val_acc: 0.9999
Epoch 11/15
20591/20591 [=====] - 141s 7ms/sample - loss: 4.3298e-06 - acc: 1.0000 - val_loss: 5.2253e-04 - val_acc: 0.9999
Epoch 12/15
20591/20591 [=====] - 141s 7ms/sample - loss: 3.0284e-06 - acc: 1.0000 - val_loss: 5.1974e-04 - val_acc: 0.9999
Epoch 13/15
20591/20591 [=====] - 141s 7ms/sample - loss: 2.1118e-06 - acc: 1.0000 - val_loss: 4.9742e-04 - val_acc: 0.9999
Epoch 14/15
20591/20591 [=====] - 141s 7ms/sample - loss: 1.4905e-06 - acc: 1.0000 - val_loss: 5.0703e-04 - val_acc: 0.9999
Epoch 15/15
20591/20591 [=====] - 142s 7ms/sample - loss: 1.0459e-06 - acc: 1.0000 - val_loss: 5.2003e-04 - val_acc: 0.9999

```

Out[32]: <tensorflow.python.keras.callbacks.History at 0x7f59071aa400>

```

In [0]: 1 test_loss, test_acc = model.evaluate(tX, tY, verbose=2)
        2

```

7172/7172 - 10s - loss: 0.9118 - acc: 0.8938

Задание 3. Примените дополнение данных (data augmentation). Как это повлияло на качество классификатора?

```

In [0]: 1 datagen = ImageDataGenerator(zoom_range=[0.5,1.0], brightness_range=[0.2,1.0])
        2 train_generator = datagen.flow(trainX, trainY, batch_size=1)
        3 test_generator = datagen.flow(testX, testY, batch_size=1)

```

```

In [0]: 1 model = Sequential([
        2     layers.Conv2D(64, kernel_size=3, activation='relu', input_shape=(28,28,1)),
        3     layers.Conv2D(32, kernel_size=5, activation='relu'),

```

```
4     layers.MaxPooling2D((2, 2)),
5     layers.Conv2D(64, (3, 3), activation='relu', padding='same'),
6     layers.MaxPooling2D((2, 2)),
7     layers.Conv2D(128, (3, 3), activation='relu', padding='same'),
8     layers.MaxPooling2D((2, 2)),
9     layers.Flatten(),
10    layers.Dense(25, activation='softmax')
11 ])
12
13 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
14
15 model.fit_generator(
16     train_generator,
17     epochs=15)
18
```

```
Epoch 1/15
27455/27455 [=====] - 270s 10ms/step - loss: 0.4762 - accuracy: 0.8441
Epoch 2/15
27455/27455 [=====] - 268s 10ms/step - loss: 0.1365 - accuracy: 0.9553
Epoch 3/15
27455/27455 [=====] - 268s 10ms/step - loss: 0.0975 - accuracy: 0.9684
Epoch 4/15
27455/27455 [=====] - 268s 10ms/step - loss: 0.0823 - accuracy: 0.9739
Epoch 5/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0748 - accuracy: 0.9768
Epoch 6/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0690 - accuracy: 0.9791
Epoch 7/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0657 - accuracy: 0.9804
Epoch 8/15
27455/27455 [=====] - 270s 10ms/step - loss: 0.0624 - accuracy: 0.9818
```

```
Epoch 9/15
27455/27455 [=====] - 268s 10ms/step - loss: 0.0594 - accuracy: 0.9827
Epoch 10/15
27455/27455 [=====] - 268s 10ms/step - loss: 0.0574 - accuracy: 0.9837
Epoch 11/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0557 - accuracy: 0.9844
Epoch 12/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0557 - accuracy: 0.9846
Epoch 13/15
27455/27455 [=====] - 266s 10ms/step - loss: 0.0546 - accuracy: 0.9851
Epoch 14/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0529 - accuracy: 0.9856
Epoch 15/15
27455/27455 [=====] - 267s 10ms/step - loss: 0.0529 - accuracy: 0.9859
```

Задание 4. Поэкспериментируйте с готовыми нейронными сетями (например, AlexNet, VGG16, Inception и т.п.), применив передаточное обучение. Как это повлияло на качество классификатора? Можно ли было обойтись без него? Какой максимальный результат удалось получить на контрольной выборке?

```
In [0]: 1 from keras.applications import MobileNet
        2
        3 base_model=MobileNet(weights='imagenet',include_top=False)
        4
        5 x=base_model.output
        6 x=GlobalAveragePooling2D()(x)
        7 x=Dense(1024,activation='relu')(x)
        8 x=Dense(1024,activation='relu')(x)
        9 x=Dense(512,activation='relu')(x)
       10 preds=Dense(25,activation='softmax')(x)
       11 model=Model(inputs=base_model.input,outputs=preds)
       12 model.summary()
```

/usr/local/lib/python3.6/dist-packages/keras_applications/mobilenet.py:207: UserWarning: `input_shape` is undefined or non-square, or `rows` is not in [128, 160, 192, 224]. Weights for input shape (224, 224) will be loaded as the default.

warnings.warn("`input_shape` is undefined or non-square, '

Model: "model_5"

Layer (type)	Output Shape	Param #
=====		
input_6 (InputLayer)	(None, None, None, 3)	0

conv1_pad (ZeroPadding2D)	(None, None, None, 3)	0

conv1 (Conv2D)	(None, None, None, 32)	864

conv1_bn (BatchNormalization)	(None, None, None, 32)	128

conv1_relu (ReLU)	(None, None, None, 32)	0

conv_dw_1 (DepthwiseConv2D)	(None, None, None, 32)	288


```
In [0]: 1 for layer in model.layers:
        2     layer.trainable=False
```

```
In [0]: 1 def transform(dataset):
        2     newDataset = list()
        3     for x in dataset:
        4         x = np.repeat(x, 3, 2)
        5         newDataset.append(x)
        6     return np.array(newDataset)
        7
        8 newTrainX = transform(trainX)
        9 newTestX = transform(testX)
```

```
In [0]: 1 print(newTrainX.shape, newTestX.shape)
```

```
(20591, 28, 28, 3) (6864, 28, 28, 3)
```

```
In [0]: 1 model.compile(optimizer='adam', loss='sparse_categorical_crossentropy', metrics=['accuracy'])
        2
        3 model.fit(newTrainX, trainY, validation_data=(newTestX, testY), epochs=15)
```

Train on 20591 samples, validate on 6864 samples

Epoch 1/15

20591/20591 [=====] - 47s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398

Epoch 2/15

20591/20591 [=====] - 44s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398

Epoch 3/15

20591/20591 [=====] - 44s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398

```
al_acc: 0.0398
Epoch 4/15
20591/20591 [=====] - 44s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 5/15
20591/20591 [=====] - 44s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 6/15
20591/20591 [=====] - 44s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 7/15
20591/20591 [=====] - 45s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 8/15
20591/20591 [=====] - 45s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 9/15
20591/20591 [=====] - 45s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 10/15
20591/20591 [=====] - 45s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 11/15
20591/20591 [=====] - 45s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 12/15
20591/20591 [=====] - 46s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 13/15
20591/20591 [=====] - 46s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 14/15
20591/20591 [=====] - 46s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - v
al_acc: 0.0398
Epoch 15/15
```

```
20591/20591 [=====] - 46s 2ms/step - loss: nan - acc: 0.0414 - val_loss: nan - val_acc: 0.0398
```

```
Out[94]: <keras.callbacks.History at 0x7f9abbce00b8>
```

Лабораторная работа №7. Рекуррентные нейронные сети для анализа текста

Данные: Набор данных для предсказания оценок для отзывов, собранных с сайта imdb.com, который состоит из 50,000 отзывов в виде текстовых файлов. Отзывы разделены на положительные (25,000) и отрицательные (25,000). Данные предварительно токенизированы по принципу “мешка слов”, индексы слов можно взять из словаря (imdb.vocab). Обучающая выборка включает в себя 12,500 положительных и 12,500 отрицательных отзывов, контрольная выборка также содержит 12,500 положительных и 12,500 отрицательных отзывов, а также. Данные можно скачать по ссылке <https://ai.stanford.edu/~amaas/data/sentiment/> (<https://ai.stanford.edu/~amaas/data/sentiment/>).

In [0]:

```
1 !pip install git+https://github.com/d2l-ai/d2l-en # installing d2l
2 !pip install -U --pre mxnet-cu101mkl # updating mxnet to at least v1.6
3
```

Collecting git+https://github.com/d2l-ai/d2l-en

Cloning https://github.com/d2l-ai/d2l-en (https://github.com/d2l-ai/d2l-en) to /tmp/pip-req-build-b7g_6gbm

Running command git clone -q https://github.com/d2l-ai/d2l-en (https://github.com/d2l-ai/d2l-en) /tmp/pip-req-build-b7g_6gbm

Requirement already satisfied: jupyter in /usr/local/lib/python3.6/dist-packages (from d2l==0.11.4) (1.0.0)

Requirement already satisfied: numpy in /usr/local/lib/python3.6/dist-packages (from d2l==0.11.4) (1.18.2)

Requirement already satisfied: matplotlib in /usr/local/lib/python3.6/dist-packages (from d2l==0.11.4) (3.2.1)

Requirement already satisfied: pandas in /usr/local/lib/python3.6/dist-packages (from d2l==0.11.4) (1.0.3)

Requirement already satisfied: jupyter-console in /usr/local/lib/python3.6/dist-packages (from jupyter->d2l==0.11.4) (5.2.0)

Requirement already satisfied: ipykernel in /usr/local/lib/python3.6/dist-packages (from jupyter->d2l==0.11.4) (4.6.1)

Requirement already satisfied: ipywidgets in /usr/local/lib/python3.6/dist-packages (from jupyter->d2l==0.11.4) (7.5.1)

Requirement already satisfied: nbconvert in /usr/local/lib/python3.6/dist-packages (from jupyter->d2l==0.11.4) (5.4.0)

```
In [0]: 1 import shutil
        2 from google.colab import drive
        3 import d2l
        4 from mxnet import gluon, np, npx, init
        5 import os
        6 from mxnet.gluon import nn, rnn
        7 from mxnet.contrib import text
        8 npx.set_np()
```

Задание 1. Загрузите данные. Преобразуйте текстовые файлы во внутренние структуры данных, которые используют индексы ВМЕСТО СЛОВ.

```
In [0]: 1 !wget https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
        2 drive.mount('/content/drive')
        3 shutil.unpack_archive("aclImdb_v1.tar.gz", "/content/aclImdb_v1/")
```

```
--2020-04-03 17:29:45-- https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz
(https://ai.stanford.edu/~amaas/data/sentiment/aclImdb_v1.tar.gz)
Resolving ai.stanford.edu (ai.stanford.edu)... 171.64.68.10
Connecting to ai.stanford.edu (ai.stanford.edu)|171.64.68.10|:443... connected.
HTTP request sent, awaiting response... 200 OK
Length: 84125825 (80M) [application/x-gzip]
Saving to: 'aclImdb_v1.tar.gz.2'
```

```
aclImdb_v1.tar.gz.2 100%[=====>] 80.23M 85.2MB/s in 0.9s
```

```
2020-04-03 17:29:46 (85.2 MB/s) - 'aclImdb_v1.tar.gz.2' saved [84125825/84125825]
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call drive.mount("/content/drive", force_remount=True).
```

```
In [0]: 1 def read_imdb(data_dir, is_train):
2         data, labels = [], []
3         for label in ('pos', 'neg'):
4             folder_name = os.path.join(data_dir, 'train' if is_train else 'test',
5                                         label)
6             for file in os.listdir(folder_name):
7                 with open(os.path.join(folder_name, file), 'rb') as f:
8                     review = f.read().decode('utf-8').replace('\n', '')
9                     data.append(review)
10                    labels.append(1 if label == 'pos' else 0)
11         return data, labels
12
13 train_data = read_imdb('/content/aclImdb_v1/aclImdb', is_train=True)
14 print('trainings:', len(train_data[0]))
```

trainings: 25000

```
In [0]: 1 train_tokens = d2l.tokenize(train_data[0], token='word')
2         vocab = d2l.Vocab(train_tokens, min_freq=5, reserved_tokens=['<pad>'])
3
4         d2l.set_figsize((3.5, 2.5))
5         d2l.plt.hist([len(line) for line in train_tokens], bins=range(0, 1000, 50))
```

<Figure size 252x180 with 1 Axes>

```
In [0]: 1 num_steps = 500 # sequence length
        2 train_features = np.array([d2l.truncate_pad(
        3     vocab[line], num_steps, vocab['<pad>']) for line in train_tokens])
        4 train_features.shape
```

Out[13]: (25000, 500)

```
In [0]: 1 train_iter = d2l.load_array((train_features, train_data[1]), 64)
        2 'batches:', len(train_iter)
```

Out[17]: ('batches:', 391)

```
In [0]: 1 def load_data_imdb(batch_size, num_steps=500):
        2     data_dir = '/content/aclImdb_v1/aclImdb'
        3     train_data = read_imdb(data_dir, True)
        4     test_data = read_imdb(data_dir, False)
        5     train_tokens = d2l.tokenize(train_data[0], token='word')
        6     test_tokens = d2l.tokenize(test_data[0], token='word')
        7     vocab = d2l.Vocab(train_tokens, min_freq=5)
        8     train_features = np.array([d2l.truncate_pad(
        9         vocab[line], num_steps, vocab.unk) for line in train_tokens])
       10     test_features = np.array([d2l.truncate_pad(
       11         vocab[line], num_steps, vocab.unk) for line in test_tokens])
       12     train_iter = d2l.load_array((train_features, train_data[1]), batch_size)
       13     test_iter = d2l.load_array((test_features, test_data[1]), batch_size,
       14         is_train=False)
       15     return train_iter, test_iter, vocab
```

Задание 2. Реализуйте и обучите двунаправленную рекуррентную сеть (LSTM или GRU). Какого качества классификации удалось достичь?

```
In [0]: 1 batch_size = 64
        2 train_iter, test_iter, vocab = load_data_imdb(batch_size)
```

```
In [0]: 1 class BiRNN(nn.Block):
        2     def __init__(self, vocab_size, embed_size, num_hiddens,
        3                 num_layers, **kwargs):
        4         super(BiRNN, self).__init__(**kwargs)
        5         self.embedding = nn.Embedding(vocab_size, embed_size)
        6         self.encoder = rnn.LSTM(num_hiddens, num_layers=num_layers,
        7                               bidirectional=True, input_size=embed_size)
        8         self.decoder = nn.Dense(2)
        9
        10    def forward(self, inputs):
        11        embeddings = self.embedding(inputs.T)
        12        outputs = self.encoder(embeddings)
        13        encoding = np.concatenate((outputs[0], outputs[-1]), axis=1)
        14        outs = self.decoder(encoding)
        15        return outs
```

```
In [0]: 1 embed_size, num_hiddens, num_layers, ctx = 100, 100, 2, d2l.try_all_gpus()
        2 net = BiRNN(len(vocab), embed_size, num_hiddens, num_layers)
        3 net.initialize(init.Xavier(), ctx=ctx)
```



```
In [0]: 1 lr, num_epochs = 0.01, 5
        2 trainer = gluon.Trainer(net.collect_params(), 'adam', {'learning_rate': lr})
        3 loss = gluon.loss.SoftmaxCrossEntropyLoss()
        4 d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, ctx)
```

```
loss 0.095, train acc 0.970, test acc 0.823
580.3 examples/sec on [gpu(0)]
```

<Figure size 252x180 with 1 Axes>

```
In [0]: 1 predict_sentiment(net, vocab, 'this movie is so great')
```

Out[54]: 'positive'

```
In [0]: 1 predict_sentiment(net, vocab, 'this movie is so bad')
```

Out[55]: 'negative'

Задание 3. Используйте индексы слов и их различное внутреннее представление (word2vec, glove). Как влияет данное преобразование на качество классификации?

```
In [0]: 1 glove_embedding = text.embedding.create('glove', pretrained_file_name='glove.6B.100d.txt')
```

```
Downloading /root/.mxnet/embeddings/glove/glove.6B.zip from https://apache-mxnet.s3-accelerate.dualstack.
amazonaws.com/gluon/embeddings/glove/glove.6B.zip... (https://apache-mxnet.s3-accelerate.dualstack.amazon
aws.com/gluon/embeddings/glove/glove.6B.zip...)
```

```
In [0]: 1 embeds = glove_embedding.get_vecs_by_tokens(vocab.idx_to_token).as_np_ndarray()
        2 embeds.shape
```

Out[42]: (49339, 100)

```
In [0]: 1 net.embedding.weight.set_data(embeds)
        2 net.embedding.collect_params().setattr('grad_req', 'null')
```

```
In [0]: 1 lr, num_epochs = 0.01, 5
        2 trainer = gluon.Trainer(net.collect_params(), 'adam', {'learning_rate': lr})
        3 loss = gluon.loss.SoftmaxCrossEntropyLoss()
        4 d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, ctx)
```

loss 0.317, train acc 0.866, test acc 0.844
604.6 examples/sec on [gpu(0)]

<Figure size 252x180 with 1 Axes>

```
In [0]: 1 def predict_sentiment(net, vocab, sentence):
        2     sentence = np.array(vocab[sentence.split()], ctx=d2l.try_gpu())
        3     label = np.argmax(net(sentence.reshape(1, -1)), axis=1)
        4     return 'positive' if label == 1 else 'negative'
```

```
In [0]: 1 predict_sentiment(net, vocab, 'this movie is so great')
```

Out[50]: 'positive'

```
In [0]: 1 predict_sentiment(net, vocab, 'this movie is so bad')
```

Out[51]: 'negative'

Задание 4. Поэкспериментируйте со структурой сети (добавьте больше рекуррентных, полносвязных или сверточных слоев). Как это повлияло на качество классификации?

```
In [0]: 1 embed_size, num_hiddens, num_layers, ctx = 100, 100, 4, d2l.try_all_gpus()
        2 net = BiRNN(len(vocab), embed_size, num_hiddens, num_layers)
        3 net.initialize(init.Xavier(), ctx=ctx)
```

```
In [0]: 1 lr, num_epochs = 0.01, 5
        2 trainer = gluon.Trainer(net.collect_params(), 'adam', {'learning_rate': lr})
        3 loss = gluon.loss.SoftmaxCrossEntropyLoss()
        4 d2l.train_ch13(net, train_iter, test_iter, loss, trainer, num_epochs, ctx)
```

loss 0.075, train acc 0.975, test acc 0.840
265.2 examples/sec on [gpu(0)]

<Figure size 252x180 with 1 Axes>

Задание 5. Используйте предобученную рекуррентную нейронную сеть (например, DeepMoji или что-то подобное). Какой максимальный результат удалось получить на контрольной выборке?

```
In [0]: 1 !git clone https://github.com/bfelbo/DeepMoji.git
```

Cloning into 'DeepMoji'...
remote: Enumerating objects: 281, done.
remote: Total 281 (delta 0), reused 0 (delta 0), pack-reused 281
Receiving objects: 100% (281/281), 110.54 MiB | 28.08 MiB/s, done.
Resolving deltas: 100% (142/142), done.
Checking out files: 100% (66/66), done.

```
In [0]: 1 !pip install tensorflow==1.14
```

Collecting tensorflow==1.14
 Downloading https://files.pythonhosted.org/packages/de/f0/96fb2e0412ae9692dbf400e5b04432885f677ad6241c088ccc5fe7724d69/tensorflow-1.14.0-cp36-cp36m-manylinux1_x86_64.whl

```
(https://files.pythonhosted.org/packages/de/f0/96fb2e0412ae9692dbf400e5b04432885f677ad6241c088ccc5fe7724d69/tensorflow-1.14.0-cp36-cp36m-manylinux1_x86_64.whl) (109.2MB)
|████████████████████████████████████████| 109.2MB 97kB/s
Requirement already satisfied: keras-preprocessing>=1.0.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.1.0)
Requirement already satisfied: astor>=0.6.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (0.8.1)
Requirement already satisfied: six>=1.10.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.12.0)
Requirement already satisfied: termcolor>=1.1.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.1.0)
Requirement already satisfied: wheel>=0.26 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (0.34.2)
Collecting tensorboard<1.15.0,>=1.14.0
  Downloading https://files.pythonhosted.org/packages/91/2d/2ed263449a078cd9c8a9ba50ebd50123adf1f8cfbea1492f9084169b89d9/tensorboard-1.14.0-py3-none-any.whl
    (https://files.pythonhosted.org/packages/91/2d/2ed263449a078cd9c8a9ba50ebd50123adf1f8cfbea1492f9084169b89d9/tensorboard-1.14.0-py3-none-any.whl) (3.1MB)
    |████████████████████████████████████████| 3.2MB 49.4MB/s
Requirement already satisfied: gast>=0.2.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (0.3.3)
Requirement already satisfied: absl-py>=0.7.0 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (0.9.0)
Requirement already satisfied: wrapt>=1.11.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.12.1)
Collecting tensorflow-estimator<1.15.0rc0,>=1.14.0rc0
  Downloading https://files.pythonhosted.org/packages/3c/d5/21860a5b11caf0678fbc8319341b0ae21a07156911132e0e71bfffed0510d/tensorflow_estimator-1.14.0-py2.py3-none-any.whl
    (https://files.pythonhosted.org/packages/3c/d5/21860a5b11caf0678fbc8319341b0ae21a07156911132e0e71bfffed0510d/tensorflow_estimator-1.14.0-py2.py3-none-any.whl) (488kB)
    |████████████████████████████████████████| 491kB 56.5MB/s
Requirement already satisfied: protobuf>=3.6.1 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (3.10.0)
Requirement already satisfied: google-pasta>=0.1.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (0.2.0)
```

```

Requirement already satisfied: grpcio>=1.8.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.27.2)
Requirement already satisfied: numpy<2.0,>=1.14.5 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.18.2)
Requirement already satisfied: keras-applications>=1.0.6 in /usr/local/lib/python3.6/dist-packages (from tensorflow==1.14) (1.0.8)
Requirement already satisfied: markdown>=2.6.8 in /usr/local/lib/python3.6/dist-packages (from tensorboard<1.15.0,>=1.14.0->tensorflow==1.14) (3.2.1)
Requirement already satisfied: werkzeug>=0.11.15 in /usr/local/lib/python3.6/dist-packages (from tensorboard<1.15.0,>=1.14.0->tensorflow==1.14) (1.0.1)
Requirement already satisfied: setuptools>=41.0.0 in /usr/local/lib/python3.6/dist-packages (from tensorboard<1.15.0,>=1.14.0->tensorflow==1.14) (46.0.0)
Requirement already satisfied: h5py in /usr/local/lib/python3.6/dist-packages (from keras-applications>=1.0.6->tensorflow==1.14) (2.10.0)
Installing collected packages: tensorboard, tensorflow-estimator, tensorflow
  Found existing installation: tensorboard 2.2.0
    Uninstalling tensorboard-2.2.0:
      Successfully uninstalled tensorboard-2.2.0
  Found existing installation: tensorflow-estimator 2.2.0rc0
    Uninstalling tensorflow-estimator-2.2.0rc0:
      Successfully uninstalled tensorflow-estimator-2.2.0rc0
  Found existing installation: tensorflow 2.2.0rc2
    Uninstalling tensorflow-2.2.0rc2:
      Successfully uninstalled tensorflow-2.2.0rc2
Successfully installed tensorboard-1.14.0 tensorflow-1.14.0 tensorflow-estimator-1.14.0

```

```

In [0]: 1 import os
        2 os.chdir('/content/DeepMoji')

```

```

In [0]: 1 from __future__ import print_function
        2 import deepmoji
        3 import examples.example_helper
        4 import numpy as np
        5 from keras.preprocessing import sequence

```

```
6 from keras.datasets import imdb
7 import deepmoji.model_def
8
9 nb_tokens = 20000
10 maxlen = 80
11 batch_size = 32
12
13 print('Loading data...')
14 print(len(train_data[0]), 'train sequences')
15 print(len(test_data[0]), 'test sequences')
16 print('Pad sequences (samples x time)')
17 print('X_train shape:', train_data.shape)
18 print('X_test shape:', test_data.shape)
19
20 print('Build model...')
21 model = deepmoji.model_def.deepmoji_architecture(nb_classes=2, nb_tokens=nb_tokens, maxlen=maxlen)
22 model.summary()
23
24 model.compile(loss='binary_crossentropy',
25               optimizer='adam',
26               metrics=['accuracy'])
27
28 print('Train...')
29 model.fit(train_data[0], train_data[1], batch_size=batch_size, epochs=5,
30           validation_data=(test_data[0], test_data[0]))
31 score, acc = model.evaluate(test_data[0], test_data[1], batch_size=batch_size)
32 print('Test score:', score)
33 print('Test accuracy:', acc)
```

Loading data...

```

25000 train sequences
25000 test sequences
Pad sequences (samples x time)
X_train shape: (25000, 80)
X_test shape: (25000, 80)
Build model...
Model: "DeepMoji"

```

Layer (type)	Output Shape	Param #	Connected to
input_2 (InputLayer)	(None, 80)	0	
embedding (Embedding)	(None, 80, 256)	5120000	input_2[0][0]
activation_2 (Activation)	(None, 80, 256)	0	embedding[0][0]
bi_lstm_0 (Bidirectional)	(None, 80, 1024)	3149824	activation_2[0][0]
bi_lstm_1 (Bidirectional)	(None, 80, 1024)	6295552	bi_lstm_0[0][0]
concatenate_2 (Concatenate)	(None, 80, 2304)	0	bi_lstm_1[0][0] bi_lstm_0[0][0] activation_2[0][0]
attlayer (AttentionWeightedAver	(None, 2304)	2304	concatenate_2[0][0]
softmax (Dense)	(None, 1)	2305	attlayer[0][0]

```

Total params: 14,569,985
Trainable params: 14,569,985
Non-trainable params: 0

```

```

Train...
Train on 25000 samples, validate on 25000 samples
Epoch 1/5

```

```
25000/25000 [=====] - 6155s 246ms/step - loss: 0.4263 - acc: 0.8038 - val_loss:
0.3644 - val_acc: 0.8446
Epoch 2/5
25000/25000 [=====] - 5949s 238ms/step - loss: 0.2494 - acc: 0.9028 - val_loss:
0.3690 - val_acc: 0.8428
Epoch 3/5
25000/25000 [=====] - 5676s 227ms/step - loss: 0.1577 - acc: 0.9453 - val_loss:
0.4109 - val_acc: 0.8388
Epoch 4/5
25000/25000 [=====] - 5633s 225ms/step - loss: 0.0913 - acc: 0.9715 - val_loss:
0.5532 - val_acc: 0.8300
Epoch 5/5
25000/25000 [=====] - 5886s 235ms/step - loss: 0.0647 - acc: 0.9823 - val_loss:
0.7103 - val_acc: 0.8306
25000/25000 [=====] - 1033s 41ms/step
Test score: 0.7102789056491852
Test accuracy: 0.83064
```

Лабораторная работа №8. Рекуррентные нейронные сети для анализа временных рядов

Данные: Набор данных для прогнозирования временных рядов, который состоит из среднемесячного числа пятен на солнце, наблюдаемых с января 1749 по август 2017. Данные в виде csv-файла можно скачать на сайте Kaggle -> <https://www.kaggle.com/robervalt/sunspots/> (<https://www.kaggle.com/robervalt/sunspots/>)

In [0]:

```
1 from google.colab import drive
2 drive.mount('/content/drive')
```

Go to this URL in a browser: https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly (https://accounts.google.com/o/oauth2/auth?client_id=947318989803-6bn6qk8qdgf4n4g3pfee6491hc0brc4i.apps.googleusercontent.com&redirect_uri=urn%3aietf%3awg%3aoauth%3a2.0%3aoob&response_type=code&scope=email%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdocs.test%20https%3a%2f%2fwww.googleapis.com%2fauth%2fdrive.photos.readonly%20https%3a%2f%2fwww.googleapis.com%2fauth%2fpeopleapi.readonly)

Enter your authorization code:

.....

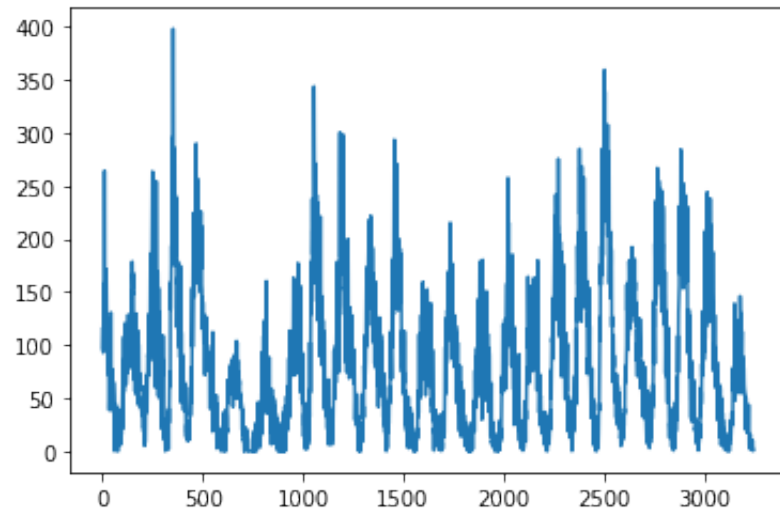
Mounted at /content/drive

```
In [0]: 1 import numpy
        2 import matplotlib.pyplot as plt
        3 import pandas
        4 import math
        5 from keras.models import Sequential
        6 from keras.layers import Dense
        7 from keras.layers import LSTM
        8 from sklearn.preprocessing import MinMaxScaler
        9 from sklearn.metrics import mean_squared_error
       10 from pandas.plotting import autocorrelation_plot
       11 from statsmodels.tsa.arima_model import ARIMA
```

```
/usr/local/lib/python3.6/dist-packages/statsmodels/tools/_testing.py:19: FutureWarning: pandas.util.testi
ng is deprecated. Use the functions in the public API at pandas.testing instead.
  import pandas.util.testing as tm
```

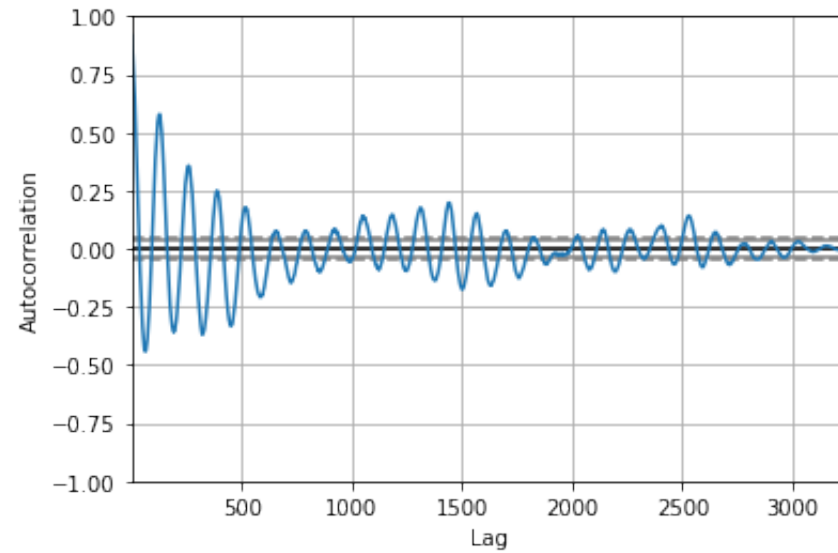
Задание 1. Загрузите данные. Изобразите ряд в виде графика. Вычислите основные характеристики временного ряда (сезонность, тренд, автокорреляцию).

```
In [0]: 1 dataframe = pandas.read_csv('/content/drive/My Drive/Collab Data/Sunspots.csv', usecols=[2], engine='py
2 plt.plot(dataframe)
3 plt.show()
```



```
In [0]: 1 autocorrelation_plot(dataframe)
```

```
Out[4]: <matplotlib.axes._subplots.AxesSubplot at 0x7fec8b4d9588>
```



```
In [0]: 1 indexes = [i for i in range(dataframe['Monthly Mean Total Sunspot Number'].count())]
2 vals = dataframe['Monthly Mean Total Sunspot Number'].values
3
4 def trendline(index,data, order=1):
5     coeffs = np.polyfit(index, list(data), order)
6     slope = coeffs[-2]
7     return float(slope)
8
9 resultent=trendline(indexes, vals)
10 print(resultent)
```

0.003319263396520065

Задание 2. Для прогнозирования разделите временной ряд на обучающую, валидационную и контрольную выборки.

```
In [0]: 1 dataset = dataframe.values
2 dataset = dataset.astype('float32')
3 scaler = MinMaxScaler(feature_range=(0, 1))
4 dataset = scaler.fit_transform(dataset)
5 print(len(dataset))
6
7 train_size = int(len(dataset) * 0.7)
8 validation_size = int(len(dataset) * 0.2)
9 test_size = len(dataset) - train_size - validation_size
10 train, validation, test = dataset[0:train_size,:], dataset[train_size:train_size+validation_size,:], dataset[train_size+validation_size:]
11 print(len(train), len(validation), len(test))
```

3249

2274 649 326

```
In [0]: 1 def create_dataset(dataset, look_back=1):
        2     dataX, dataY = [], []
        3     for i in range(len(dataset)-look_back-1):
        4         a = dataset[i:(i+look_back), 0]
        5         dataX.append(a)
        6         dataY.append(dataset[i + look_back, 0])
        7     return numpy.array(dataX), numpy.array(dataY)
```

```
In [0]: 1 trainX, trainY = create_dataset(train)
        2 validationX, validationY = create_dataset(validation)
        3 testX, testY = create_dataset(test)
```

```
In [0]: 1 trainX = numpy.reshape(trainX, (trainX.shape[0], 1, trainX.shape[1]))
        2 validationX = numpy.reshape(validationX, (validationX.shape[0], 1, validationX.shape[1]))
        3 testX = numpy.reshape(testX, (testX.shape[0], 1, testX.shape[1]))
```

Задание 3. Примените модель ARIMA для прогнозирования значений данного временного ряда.

```
In [0]: 1 model = ARIMA(dataframe, order=(5,1,0))
        2 model_fit = model.fit(dis=0)
        3 print(model_fit.summary())
        4
        5 residuals = pandas.DataFrame(model_fit.resid)
        6 residuals.plot()
        7 plt.show()
        8 residuals.plot(kind='kde')
        9 plt.show()
       10 print(residuals.describe())
```

ARIMA Model Results

```

=====
Dep. Variable:      D.Monthly Mean Total Sunspot Number    No. Observations:      3248
Model:              ARIMA(5, 1, 0)                        Log Likelihood          -15090.428
Method:             css-mle                               S.D. of innovations     25.206
Date:              Mon, 06 Apr 2020                       AIC                     30194.857
Time:              19:08:43                               BIC                     30237.457
Sample:            1                                       HQIC                    30210.118
=====

```

```

=====
=====
coef      std err      z      P>|z|      [0.025      0.
975]
-----
----
const      -0.0316      0.213      -0.148      0.882      -0.450      0
.386
ar.L1.D.Monthly Mean Total Sunspot Number -0.4152      0.018      -23.708      0.000      -0.450      -0
.381
ar.L2.D.Monthly Mean Total Sunspot Number -0.2979      0.019      -15.770      0.000      -0.335      -0
.261
ar.L3.D.Monthly Mean Total Sunspot Number -0.2011      0.019      -10.434      0.000      -0.239      -0
.163
ar.L4.D.Monthly Mean Total Sunspot Number -0.0987      0.019      -5.227      0.000      -0.136      -0
.062
ar.L5.D.Monthly Mean Total Sunspot Number -0.0613      0.018      -3.502      0.000      -0.096      -0
.027

```

Roots

```

=====
Real      Imaginary      Modulus      Frequency
-----
AR.1      0.8392      -1.4348j      1.6622      -0.1658
AR.2      0.8392      +1.4348j      1.6622      0.1658
AR.3      -1.7636      -0.0000j      1.7636      -0.5000
AR.4      -0.7624      -1.6629j      1.8293      -0.3184

```

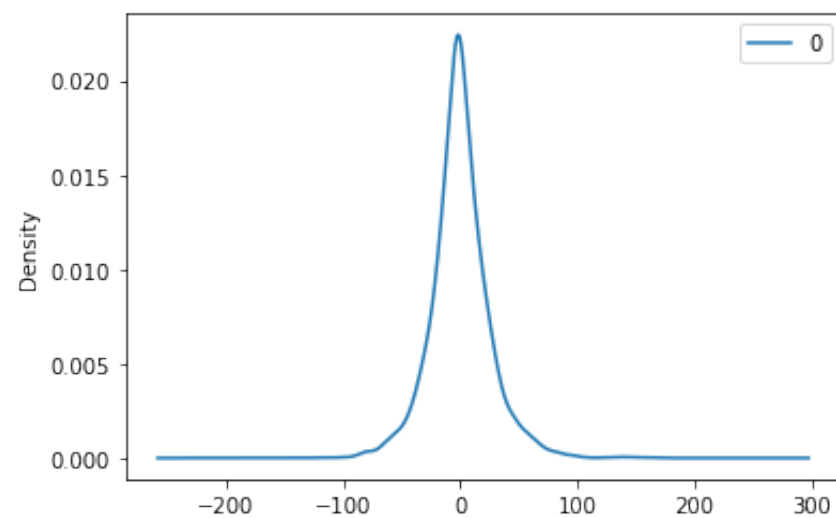
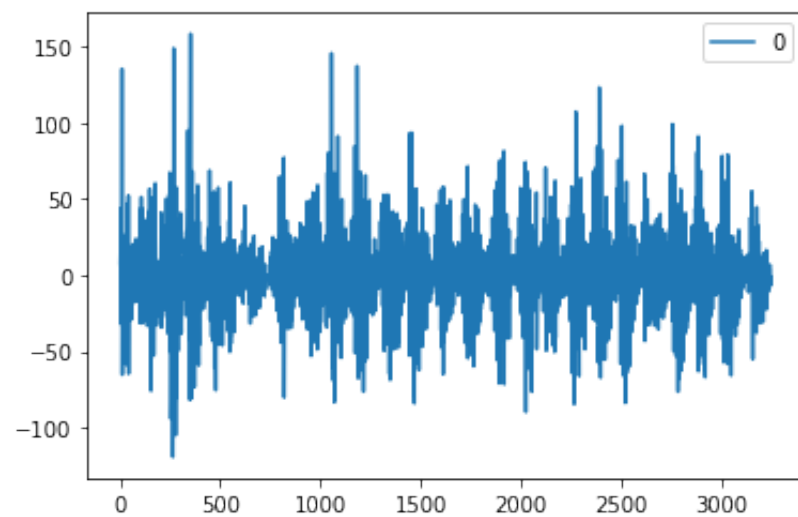
AR.5

-0.7624

+1.6629j

1.8293

0.3184



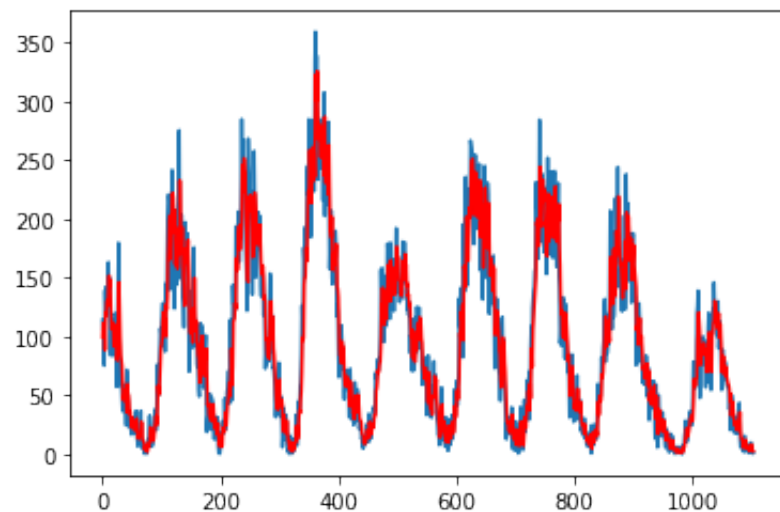
0
count 3248.000000


```
mean      0.003869
std       25.210241
min      -119.111598
25%      -13.129759
50%       -0.905605
75%       12.498157
max       158.109875
```

```
In [0]: 1 from sklearn.metrics import mean_squared_error
        2
        3 X = dataframe.values
        4 size = int(len(X) * 0.66)
        5 train, test = X[0:size], X[size:len(X)]
        6 history = [x for x in train]
        7 predictions = list()
        8 for t in tqdm(range(len(test))):
        9     model = ARIMA(history, order=(5,1,0))
       10     model_fit = model.fit(dispatch=0)
       11     output = model_fit.forecast()
       12     yhat = output[0]
       13     predictions.append(yhat)
       14     obs = test[t]
       15     history.append(obs)
       16     # print('predicted=%f, expected=%f' % (yhat, obs))
       17 error = mean_squared_error(test, predictions)
       18 print('Test MSE: %.3f' % error)
       19 # plot
       20 plt.plot(test)
       21 plt.plot(predictions, color='red')
       22 plt.show()
```

Test MSE: 623.160

Рисунок 1. 020.100



Задание 4. Повторите эксперимент по прогнозированию, реализовав рекуррентную нейронную сеть (с как минимум 2 рекуррентными слоями).

Задание 5. Сравните качество прогноза моделей. Какой максимальный результат удалось получить на контрольной выборке?

```
In [0]: 1 batch_size = 1
        2 look_back = 1
        3
        4 model = Sequential()
        5 model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True, return_sequences=True))
        6 model.add(LSTM(4, batch_input_shape=(batch_size, look_back, 1), stateful=True))
        7 model.add(Dense(1))
        8
        9
        10 model.summary()
        11
        12 model.compile(loss='mean_squared_error', optimizer='adam')
```

```
13 model.fit(trainX, trainY, epochs=50, batch_size=1, verbose=2, validation_data=(validationX, validationY))
```

Model: "sequential_18"

Layer (type)	Output Shape	Param #
lstm_26 (LSTM)	(1, 1, 4)	96
lstm_27 (LSTM)	(1, 4)	144
dense_12 (Dense)	(1, 1)	5

Total params: 245

Trainable params: 245

Non-trainable params: 0

Train on 2272 samples, validate on 647 samples

Epoch 1/50

- 8s - loss: 0.0257 - val_loss: 0.0134

Epoch 2/50

- 6s - loss: 0.0182 - val_loss: 0.0613

Epoch 3/50

In [0]:

```
1 testPredict = model.predict(testX)
2 testPredict = scaler.inverse_transform(testPredict)
3 testY = scaler.inverse_transform([testY])
4 testScore = math.sqrt(mean_squared_error(testY[0], testPredict[:,0]))
5 print('Test Score: %.2f RMSE' % (testScore))
```

Test Score: 52.32 RMSE

