

Министерство образования Республики Беларусь  
Учреждение образования  
«Белорусский государственный университет информатики и радиоэлектроники»  
Факультет компьютерных систем и сетей  
Кафедра информатики

# Машинное обучение

## Лабораторные работы №№1-8

**Выполнила:**  
магистрант специальности ИиТРПО  
Евтушенко Елизавета Юрьевна  
2 курс, группа 858341

**Проверил:**  
Стержанов Максим Валерьевич

# Лабораторная работа №1

## Логистическая регрессия в качестве нейронной сети

В работе предлагается использовать набор данных *notMNIST*, который состоит из изображений размерностью  $28 \times 28$  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) ([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) ([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>).

### Задание 1

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

In [0]:

```
import warnings  
  
warnings.filterwarnings('ignore')
```

In [0]:

```
SMALL_DS_URL = (  
    'https://commondatastorage.googleapis.com/books1000/notMNIST_small.tar.gz')  
LARGE_DS_URL = (  
    'https://commondatastorage.googleapis.com/books1000/notMNIST_large.tar.gz')
```

In [0]:

```
%matplotlib inline  
  
import matplotlib.pyplot as plt  
import seaborn as sns  
from matplotlib import rcParams  
  
rcParams['figure.figsize'] = 8, 6  
  
sns.set()  
sns.set_palette(sns.color_palette('hls'))
```

In [0]:

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

def tar_to_dir(_tar_url, _key):
    dir_name_ = 'dataset_' + _key
    local_file_name_ = dir_name_ + '.f'

    urlretrieve(_tar_url, local_file_name_)

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

    os.remove(local_file_name_)

    return dir_name_
```

In [0]:

```
def get_examples(_dataframe, _label_column_name, _data_column_name):
    n_ = _dataframe[_label_column_name].nunique()

    examples_ = _dataframe.sample(n_)[_data_column_name]

    return examples_
```

In [0]:

```
from math import ceil
import numpy as np

def print_examples(_examples):
    fig = plt.figure(figsize = (8, 4))

    height_ = 2
    width_ = ceil(_examples.count() / height_)

    for i, item_ in enumerate(_examples):

        ax = fig.add_subplot(height_, width_, i + 1)
        ax.axis('off')
        ax.imshow(item_, cmap = 'gray', interpolation = 'none')

    plt.show()
```

In [0]:

```
from imageio import imread
import pandas as pd

def image_to_array(_image):
    try:
        array_ = imread(_image)

        return True, array_
    except:
        return False, None

def get_inner_dir(_dir_path):
    return [x[0] for x in os.walk(_dir_path)][1]

def remove_duplicates(_dataframe, _data_column_name):
    return (_dataframe
            .loc[_dataframe[_data_column_name]
                  .astype(str).drop_duplicates().index])

def dir_to_dataframe(_dir_path):
    dataframes_ = []
    inner_dir_path_ = get_inner_dir(_dir_path)
    for subdir_ in sorted(os.listdir(inner_dir_path_)):
        letter_ = subdir_
        data_ = []
        files_ = os.listdir(os.path.join(inner_dir_path_, subdir_))
        for f in files_:
            file_path_ = os.path.join(inner_dir_path_, subdir_, f)
            can_read_, im = image_to_array(file_path_)

            if can_read_:
                data_.append(im)

        g = [letter_] * len(data_)
        e = np.array(data_)
        h = pd.DataFrame()
        h['data'] = data_
        h['label'] = letter_
        dataframes_.append(h)

    result_ = pd.concat(dataframes_, ignore_index = True)
    unique_ = remove_duplicates(result_, 'data')

    return unique_
```

```
In [0]:
```

```
def tar_to_dataframe(_tar_url, _key):  
    dir_name_ = tar_to_dir(_tar_url, _key)  
    inner_dir_ = get_inner_dir(dir_name_)  
    dataframe_ = dir_to_dataframe(dir_name_)  
    examples_ = get_examples(dataframe_, 'label', 'data')  
    print_examples(examples_)  
  
    return dataframe_
```

```
In [9]:
```

```
small_dataframe = tar_to_dataframe(SMALL_DS_URL, 'small')
```



```
In [10]:
```

```
large_dataframe = tar_to_dataframe(LARGE_DS_URL, 'large')
```



## Задание 2

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

In [0]:

```
def print_balance(_dataframe, _label_column_name):  
    values_ = (_dataframe[_label_column_name]  
               .value_counts().sort_values(ascending = False))  
  
    print((':>10') * len(values_)).format(*values_))
```

In [12]:

```
print_balance(small_dataframe, 'label')
```

1853	1850	1850	1848	1848	1848	1847
1847	1845	1596				

In [13]:

```
print_balance(large_dataframe, 'label')
```

47226	47102	47012	46890	46771	46663	46577	4
6521	46098	41086					

Как видим, классы сбалансированы.

## Задание 3

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

In [0]:

```
def split(_dataframe, _n_train, _n_test, _n_val):  
  
    assert _dataframe.shape[0] >= _n_train + _n_test + _n_val  
  
    to_be_split_ = _dataframe.copy(deep = True)  
  
    seed_ = 666  
  
    train_ = to_be_split_.sample(n = _n_train, random_state = seed_)  
  
    to_be_split_ = to_be_split_.drop(train_.index)  
    test_ = to_be_split_.sample(n = _n_test, random_state = seed_)  
  
    val_ = (to_be_split_  
           .drop(test_.index).sample(n = _n_val, random_state = seed_))  
  
    return train_, test_, val_
```

In [15]:

```
large_dataframe.shape[0]
```

Out[15]:

```
461946
```

In [16]:

```
train, test, validation = split(large_dataframe, 200000, 10000, 19000)

print_balance(train, 'label')
print_balance(test, 'label')
print_balance(validation, 'label')
```

	20415	20350	20317	20290	20278	20191	20170	2
0124	20100	17765						
	1049	1043	1033	1029	1021	1016	995	
981	961	872						
	2014	1945	1934	1931	1912	1903	1898	
1887	1864	1712						

Видно, что удалось сохранить баланс между классами.

## 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]:

```
def no_duplicates(_dataframe, _data_column_name):

    original_length_ = _dataframe.shape[0]

    unique_length_ = (_dataframe[_data_column_name]
                      .astype(str).unique().shape[0])

    print(str(original_length_) + ' -- ' + str(unique_length_))

    return original_length_ == unique_length_
```

In [18]:

```
print(no_duplicates(small_dataframe, 'data'))
```

```
18232 -- 18232
True
```

In [19]:

```
print(no_duplicates(large_dataframe, 'data'))
```

```
461946 -- 461946
True
```

In [0]:

```
small_dataframe.to_pickle("./small.pkl")
large_dataframe.to_pickle("./large.pkl")
```

Дубликатов не обнаружено, так как они были удалены на шаге построения датасета из файлов.

## 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]:

```
def dataframe_to_x_y(_dataframe):
    x_ = np.stack(_dataframe['data']).reshape(_dataframe.shape[0], -1)
    y_ = _dataframe['label'].to_numpy()
    return x_, y_
```

In [0]:

```
X_train, y_train = dataframe_to_x_y(train)
X_test, y_test = dataframe_to_x_y(test)
```

In [0]:

```
sizes = [50, 100, 1000, 50000]
clfs = {}
scores = {}
```

In [0]:

```
from sklearn.linear_model import LogisticRegression
for size_ in sizes:
    clf_ = (LogisticRegression(max_iter = 100)
            .fit(X_train[:size_], y_train[:size_]))
    clfs[size_] = clf_

```

In [25]:

```
print(*clfs[50000].predict(X_test[:10]), sep = '\t')
```

I C B H C G B E G I

In [26]:

```
print(*y_test[:10], sep = '\t')
```

C C B H C G B E G H

In [0]:

```
for size_ in sizes:  
    scores[size_] = clfs[size_].score(X_test, y_test)
```

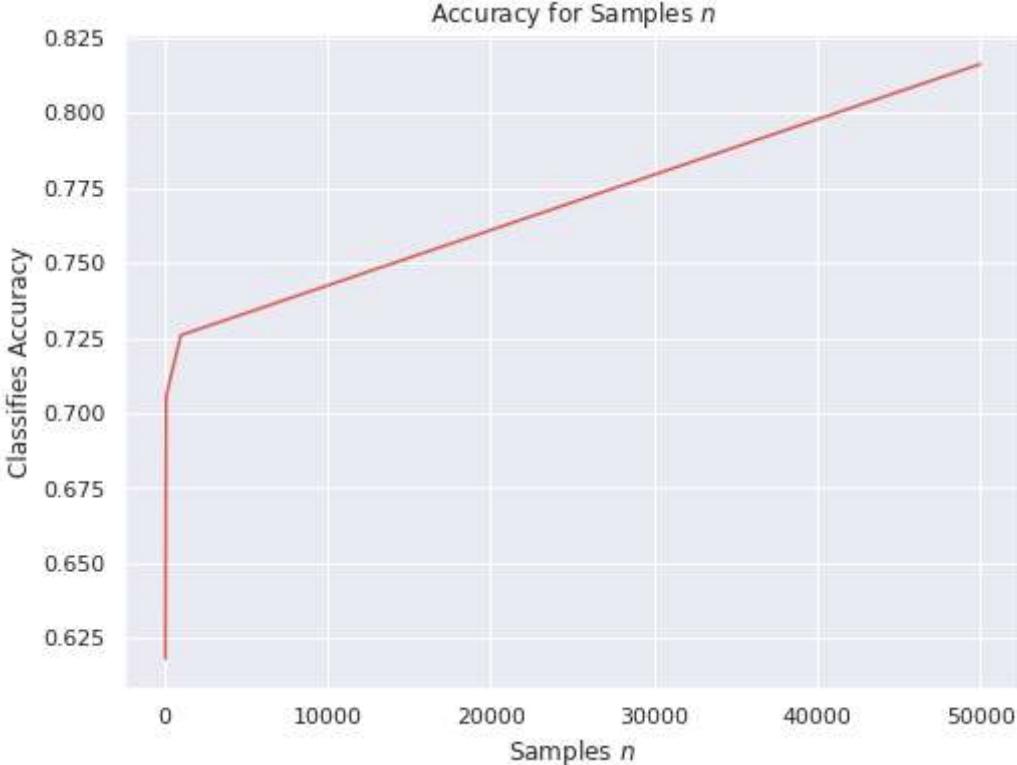
In [28]:

```
print(scores)
```

{50: 0.6183, 100: 0.7057, 1000: 0.7259, 50000: 0.8161}

In [29]:

```
sns.lineplot(sizes, [scores[s] for s in sizes])  
  
plt.xlabel('Samples $n$')  
plt.ylabel('Classifies Accuracy')  
  
plt.title('Accuracy for Samples $n$')  
  
plt.show()
```



На графике видим, что с увеличением выборки качество классификации растёт с размером выборки.



# Лабораторная работа №2

## Реализация глубокой нейронной сети

В работе предлагается использовать набор данных *notMNIST*, который состоит из изображений размерностью  $28 \times 28$  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) ([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) ([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>).

### Задание 1

Реализуйте полносвязную нейронную сеть с помощью библиотеки *TensorFlow*. 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.).

Загрузим файл с датасетом, обработанным в лабораторной работе №1.

In [0]:

```
import warnings  
  
warnings.filterwarnings('ignore')
```

In [2]:

```
from google.colab import drive  
  
drive.mount('/content/drive', force_remount = True)
```

Mounted at /content/drive

In [0]:

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

import sys
sys.path.append(BASE_DIR)

import os
os.chdir(BASE_DIR)
```

In [0]:

```
import pandas as pd

dataframe = pd.read_pickle("./large.pkl")
```

In [5]:

```
dataframe['data'].shape
```

Out[5]:

```
(461946,)
```

In [0]:

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

In [0]:

```
import tensorflow as tf
```

In [0]:

```
import numpy as np
```

In [0]:

```
dataframe_test = dataframe.sample(frac = 0.1)
dataframe = dataframe.drop(dataframe_test.index)
```

In [10]:

```
x = np.asarray(list(dataframe['data']))[..., np.newaxis]
x = tf.keras.utils.normalize(x, axis = 1)
x.shape
```

Out[10]:

```
(415751, 28, 28, 1)
```

In [11]:

```
x_test = np.asarray(list(dataframe_test['data']))[..., np.newaxis]
x_test = tf.keras.utils.normalize(x_test, axis = 1)
x_test.shape
```

Out[11]:

(46195, 28, 28, 1)

In [0]:

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

In [13]:

```
from tensorflow.keras.utils import to_categorical
y = (to_categorical(dataframe['label']
                     .astype('category').cat.codes.astype('int32')))

y.shape
```

Out[13]:

(415751, 10)

In [14]:

```
y_test = (to_categorical(dataframe_test['label']
                           .astype('category').cat.codes.astype('int32')))

y_test.shape
```

Out[14]:

(46195, 10)

In [0]:

```
LAYER_WIDTH = 5000
```

In [0]:

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

In [0]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Dense, Reshape

model = tf.keras.Sequential()

model.add(Reshape((IMAGE_DIM_0 * IMAGE_DIM_1,),
                  input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)))
model.add(Dense(LAYER_WIDTH, activation = 'relu'))
model.add(Dense(LAYER_WIDTH, activation = 'sigmoid'))
model.add(Dense(LAYER_WIDTH, activation = 'tanh'))
model.add(Dense(LAYER_WIDTH, activation = 'elu'))
model.add(Dense(LAYER_WIDTH, activation = 'softmax'))
model.add(Dense(CLASSES_N))
```

In [0]:

```
def cat_cross_from_logits(y_true, y_pred):
    return tf.keras.losses.categorical_crossentropy(
        y_true, y_pred, from_logits = True)

model.compile(optimizer = 'sgd',
              loss = cat_cross_from_logits,
              metrics = ['categorical_accuracy'])
```

In [19]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
reshape (Reshape)	(None, 784)	0
dense (Dense)	(None, 5000)	3925000
dense_1 (Dense)	(None, 5000)	25005000
dense_2 (Dense)	(None, 5000)	25005000
dense_3 (Dense)	(None, 5000)	25005000
dense_4 (Dense)	(None, 5000)	25005000
dense_5 (Dense)	(None, 10)	50010
=====		
Total params: 103,995,010		
Trainable params: 103,995,010		
Non-trainable params: 0		

In [0]:

```
VAL_SPLIT_RATE = 0.1
```

In [0]:

```
EPOCHS_N = 10
```

In [0]:

```
history = model.fit(x = x, y = y, epochs = EPOCHS_N,
                      validation_split = VAL_SPLIT_RATE, verbose = 0)
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_accuracy(_history,
                  _train_acc_name = 'accuracy',
                  _val_acc_name = 'val_accuracy'):

    plt.plot(_history.history[_train_acc_name])
    plt.plot(_history.history[_val_acc_name])

    plt.title('Model accuracy')

    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

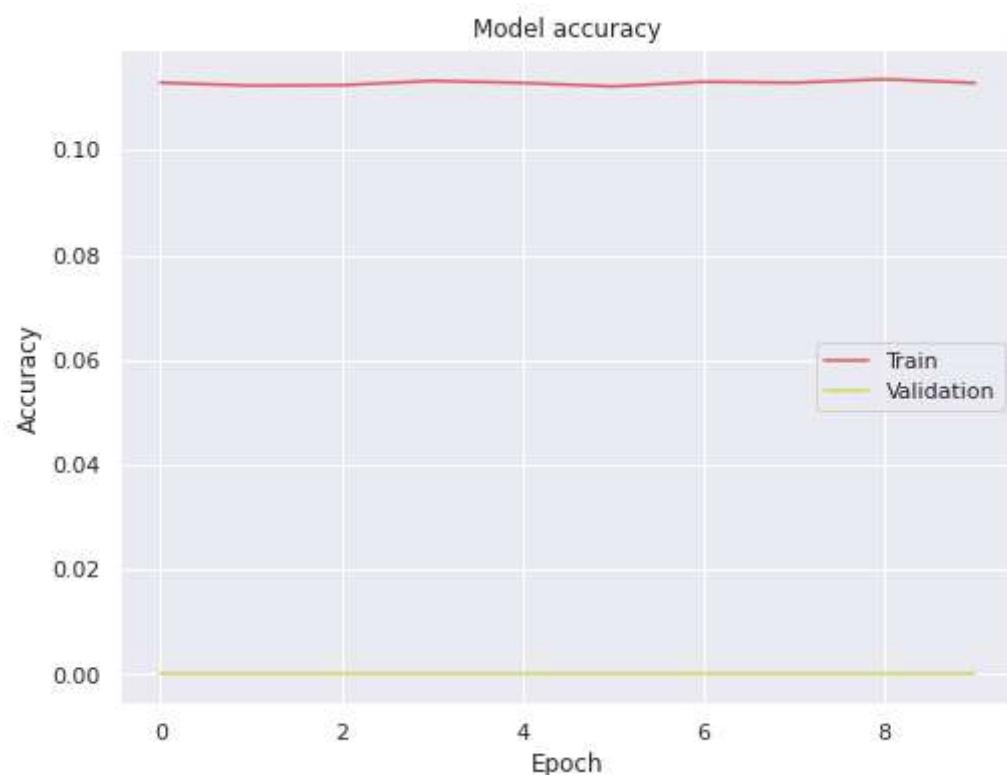
    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

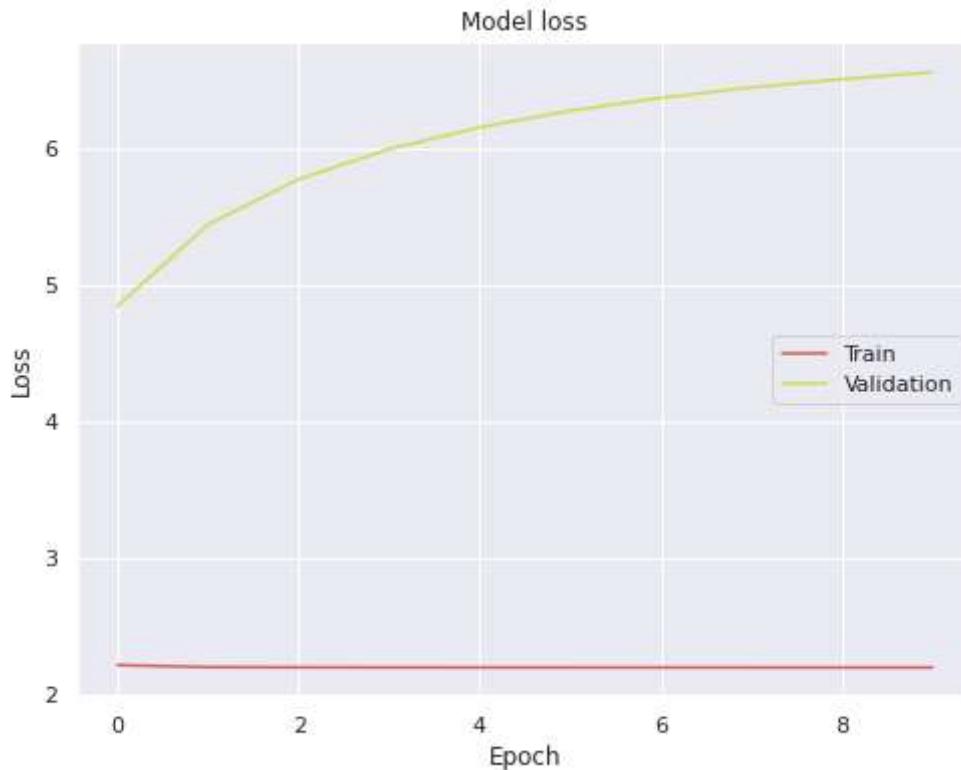
In [24]:

```
plot_accuracy(history, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [25]:

```
plot_loss(history)
```



In [26]:

```
results = model.evaluate(x_test, y_test)  
print('Test loss, test accuracy:', results)
```

```
1444/1444 [=====] - 5s 3ms/step - loss: 2.6277 - categorical_accuracy: 0.1022  
Test loss, test accuracy: [2.627650260925293, 0.10221885144710541]
```

## Задание 2

Как улучшилась точность классификатора по сравнению с логистической регрессией?

Стало хуже — на тестовой выборке точность составила 10%. Похоже, что данная модель совершенно не подходит для решения этой задачи.

## Задание 3

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

In [0]:

```
REG_RATE = 0.001
```

In [0]:

```
from tensorflow.keras.regularizers import l2  
  
l2_reg = l2(REG_RATE)
```

In [0]:

```
DROPOUT_RATE = 0.2
```

In [0]:

```
from tensorflow.keras.layers import Dropout  
  
dropout_layer = Dropout(DROPOUT_RATE)
```

In [0]:

```
model_2 = tf.keras.Sequential()  
  
model_2.add(Reshape((IMAGE_DIM_0 * IMAGE_DIM_1,),  
                    input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)))  
model_2.add(Dense(LAYER_WIDTH, activation = 'relu',  
                 kernel_regularizer = l2_reg))  
model_2.add(dropout_layer)  
model_2.add(Dense(LAYER_WIDTH, activation = 'sigmoid',  
                 kernel_regularizer = l2_reg))  
model_2.add(dropout_layer)  
model_2.add(Dense(LAYER_WIDTH, activation = 'tanh',  
                 kernel_regularizer = l2_reg))  
model_2.add(dropout_layer)  
model_2.add(Dense(LAYER_WIDTH, activation = 'sigmoid',  
                 kernel_regularizer = l2_reg))  
model_2.add(dropout_layer)  
model_2.add(Dense(LAYER_WIDTH, activation = 'relu',  
                 kernel_regularizer = l2_reg))  
model_2.add(dropout_layer)  
model_2.add(Dense(CLASSES_N))
```

In [0]:

```
model_2.compile(optimizer = 'sgd',  
                 loss = cat_crossentropy_from_logits,  
                 metrics = ['categorical_accuracy'])
```

In [33]:

```
model_2.summary()
```

Model: "sequential\_1"

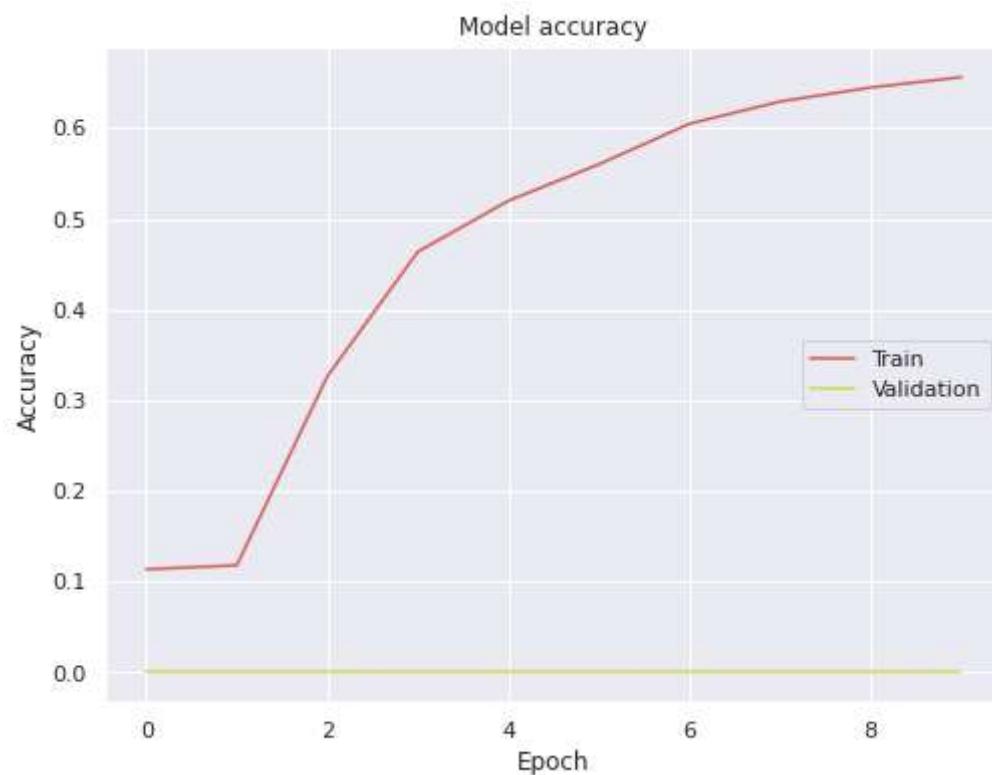
Layer (type)	Output Shape	Param #
=====		
reshape_1 (Reshape)	(None, 784)	0
dense_6 (Dense)	(None, 5000)	3925000
dropout (Dropout)	(None, 5000)	0
dense_7 (Dense)	(None, 5000)	25005000
dense_8 (Dense)	(None, 5000)	25005000
dense_9 (Dense)	(None, 5000)	25005000
dense_10 (Dense)	(None, 5000)	25005000
dense_11 (Dense)	(None, 10)	50010
=====		
Total params:	103,995,010	
Trainable params:	103,995,010	
Non-trainable params:	0	

In [0]:

```
history_2 = model_2.fit(x = x, y = y, epochs = EPOCHS_N,
                         validation_split = VAL_SPLIT_RATE, verbose = 0)
```

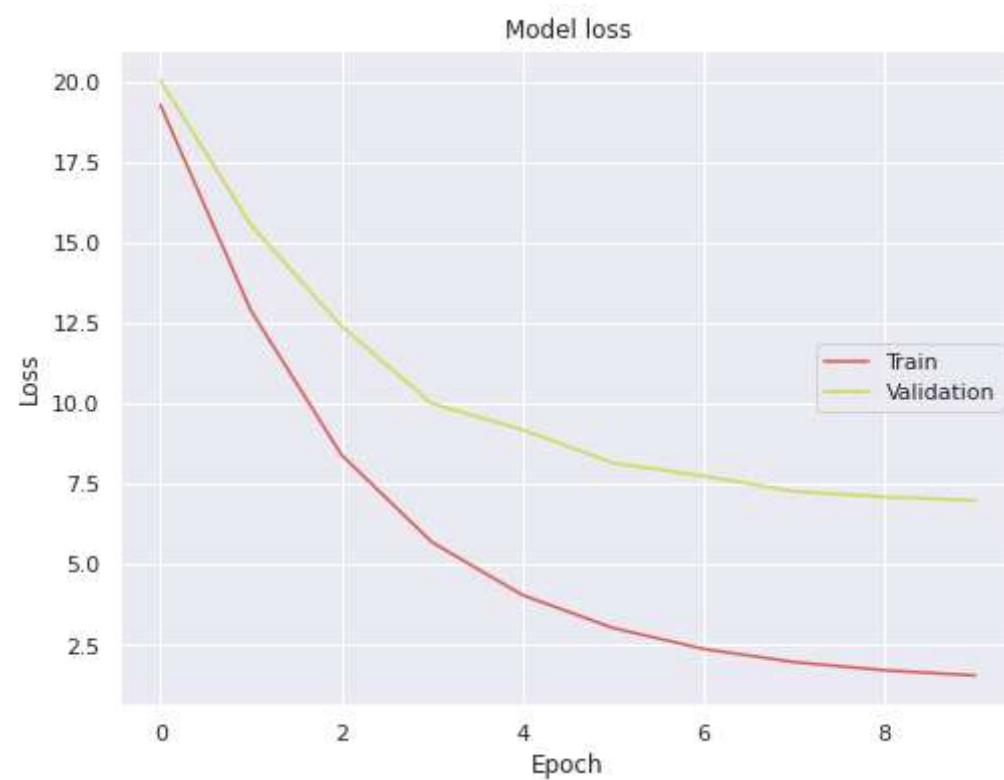
In [35]:

```
plot_accuracy(history_2, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [36]:

```
plot_loss(history_2)
```



In [37]:

```
results_2 = model_2.evaluate(x_test, y_test)

print('Test loss, test accuracy:', results_2)
```

```
1444/1444 [=====] - 8s 6ms/step - loss: 1.9703 - categorical_accuracy: 0.6346
Test loss, test accuracy: [1.9703369140625, 0.634570837020874]
```

Регуляризация и сброс нейронов значительно помогли — модель показывает 63% точности на тестовой выборке.

## 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]:

```
from tensorflow.keras.optimizers import SGD

dyn_lr_sgd = SGD(lr = 0.01, momentum = 0.9)

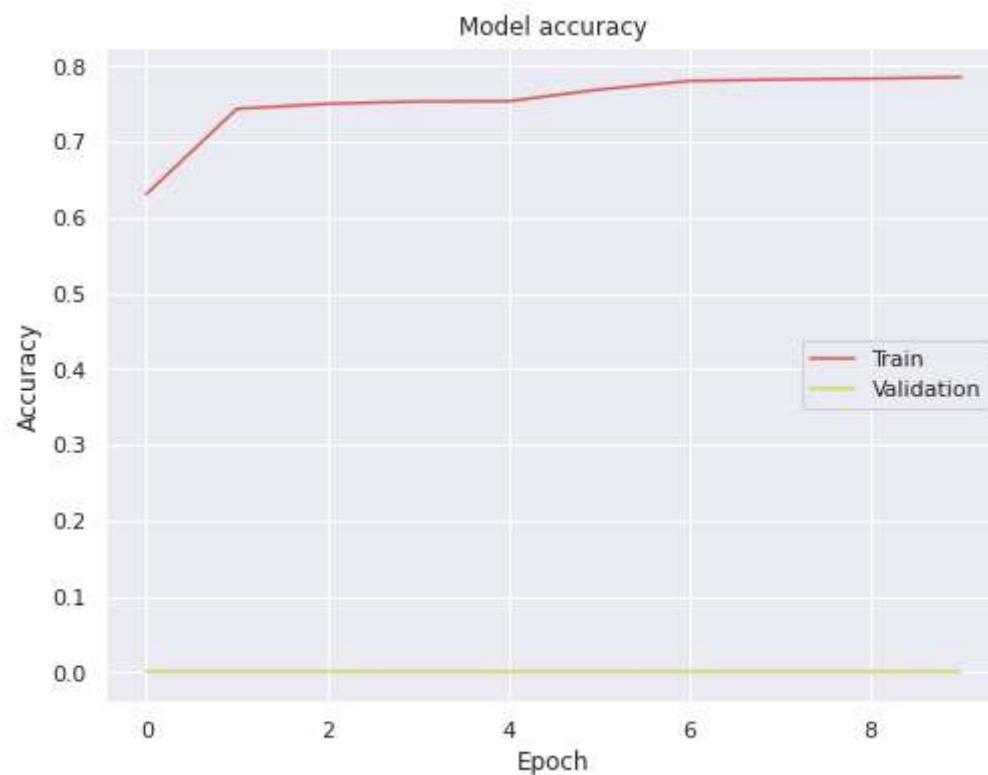
model_2.compile(optimizer = dyn_lr_sgd,
                 loss = cat_crossentropy,
                 metrics = ['categorical_accuracy'])
```

In [0]:

```
history_3 = model_2.fit(x = x, y = y, epochs = EPOCHS_N,
                        validation_split = VAL_SPLIT_RATE, verbose = 0)
```

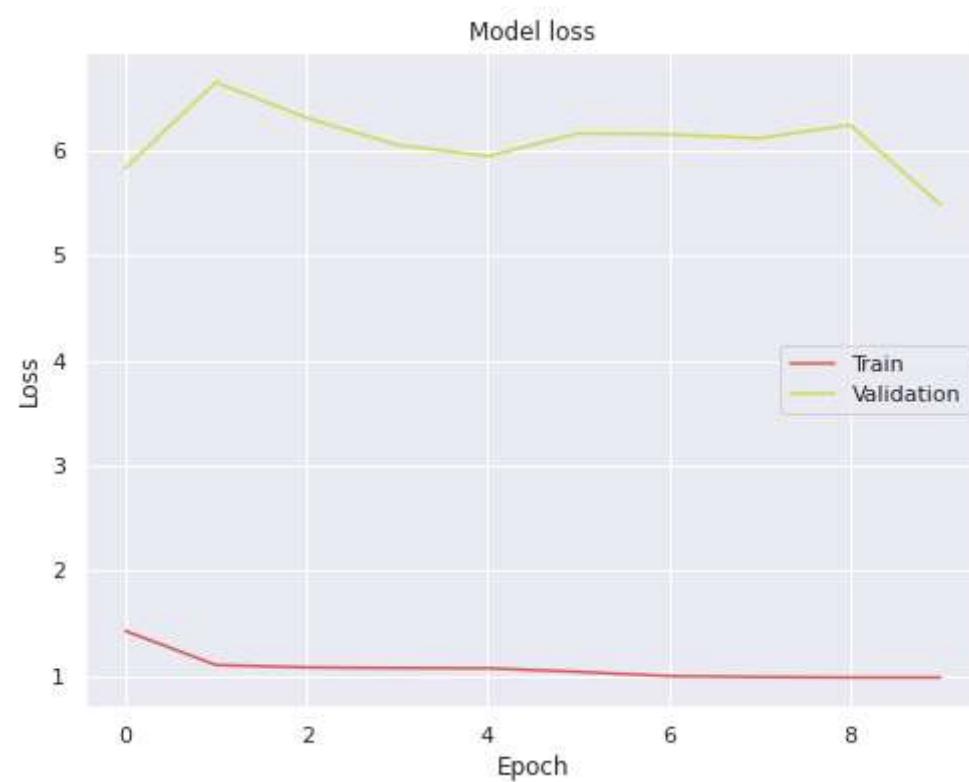
In [40]:

```
plot_accuracy(history_3, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [41]:

```
plot_loss(history_3)
```



In [42]:

```
results_3 = model_2.evaluate(x_test, y_test)
print('Test loss, test accuracy:', results_3)
```

```
1444/1444 [=====] - 8s 6ms/step - loss: 1.5546 - categorical_accuracy: 0.6496
Test loss, test accuracy: [1.5545647144317627, 0.6495941281318665]
```

Динамически изменяемая скорость обучения совсем немного улучшила результат — 64% на тестовой выборке.

Можно сделать вывод, что модель с полносвязными слоями может использоваться для решения задачи распознавания изображений, однако она очевидно не является наилучшей.

# Лабораторная работа №3

## Реализация сверточной нейронной сети

В работе предлагается использовать набор данных *notMNIST*, который состоит из изображений размерностью  $28 \times 28$  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)  
([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)  
([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>).

### Задание 1

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

Загрузим файл с датасетом, обработанным в лабораторной работе №1.

In [0]:

```
import warnings  
  
warnings.filterwarnings('ignore')
```

In [2]:

```
from google.colab import drive  
  
drive.mount('/content/drive', force_remount = True)
```

Mounted at /content/drive

In [0]:

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

import sys
sys.path.append(BASE_DIR)

import os
os.chdir(BASE_DIR)
```

In [0]:

```
import pandas as pd

dataframe = pd.read_pickle("./large.pkl")
```

In [0]:

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

In [0]:

```
import tensorflow as tf
```

In [0]:

```
# To fix memory leak: https://github.com/tensorflow/tensorflow/issues/33009

tf.compat.v1.disable_eager_execution()
```

In [0]:

```
import numpy as np
```

In [0]:

```
dataframe_test = dataframe.sample(frac = 0.1)

dataframe = dataframe.drop(dataframe_test.index)
```

In [10]:

```
x = np.asarray(list(dataframe['data']))[..., np.newaxis]
x = tf.keras.utils.normalize(x, axis = 1)
x.shape
```

Out[10]:

(415751, 28, 28, 1)

In [11]:

```
x_test = np.asarray(list(dataframe_test['data']))[..., np.newaxis]
x_test = tf.keras.utils.normalize(x_test, axis = 1)
x_test.shape
```

Out[11]:

(46195, 28, 28, 1)

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_accuracy(_history,
                  _train_acc_name = 'accuracy',
                  _val_acc_name = 'val_accuracy'):

    plt.plot(_history.history[_train_acc_name])
    plt.plot(_history.history[_val_acc_name])

    plt.title('Model accuracy')

    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

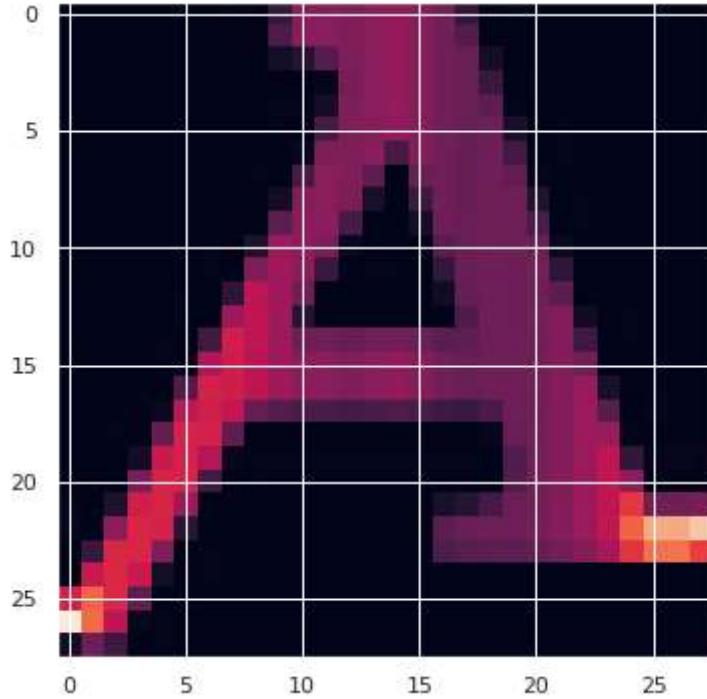
    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

In [13]:

```
plt.imshow(x[0].squeeze())  
plt.show()
```



In [0]:

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

In [15]:

```
from tensorflow.keras.utils import to_categorical  
  
y = to_categorical(dataframe['label']  
                    .astype('category').cat.codes.astype('int32'))  
  
y.shape
```

Out[15]:

```
(415751, 10)
```

In [16]:

```
y_test = to_categorical(dataframe_test['label']
                        .astype('category').cat.codes.astype('int32'))  
  
y_test.shape
```

Out[16]:

```
(46195, 10)
```

In [0]:

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

In [0]:

```
DENSE_LAYER_WIDTH = 5000
```

In [19]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Conv2D, Dense, Flatten  
  
model = tf.keras.Sequential()  
  
model.add(Conv2D(16, 3, padding = 'same', activation = 'relu',
                 input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)))
model.add(Conv2D(32, 3, padding = 'same', activation = 'relu'))
model.add(Flatten())
model.add(Dense(DENSE_LAYER_WIDTH, activation = 'relu'))
model.add(Dense(CLASSES_N))
```

WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/python/ops/resource\_variable\_ops.py:1666: 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.

In [0]:

```
def cat_cross_from_logits(y_true, y_pred):
    return tf.keras.losses.categorical_crossentropy(
        y_true, y_pred, from_logits = True)  
  
model.compile(optimizer = 'sgd',
                 loss = cat_cross_from_logits,
                 metrics = ['categorical_accuracy'])
```

In [21]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 16)	160
conv2d_1 (Conv2D)	(None, 28, 28, 32)	4640
flatten (Flatten)	(None, 25088)	0
dense (Dense)	(None, 5000)	125445000
dense_1 (Dense)	(None, 10)	50010
=====		
Total params:	125,499,810	
Trainable params:	125,499,810	
Non-trainable params:	0	

In [0]:

```
VAL_SPLIT_RATE = 0.1
```

In [0]:

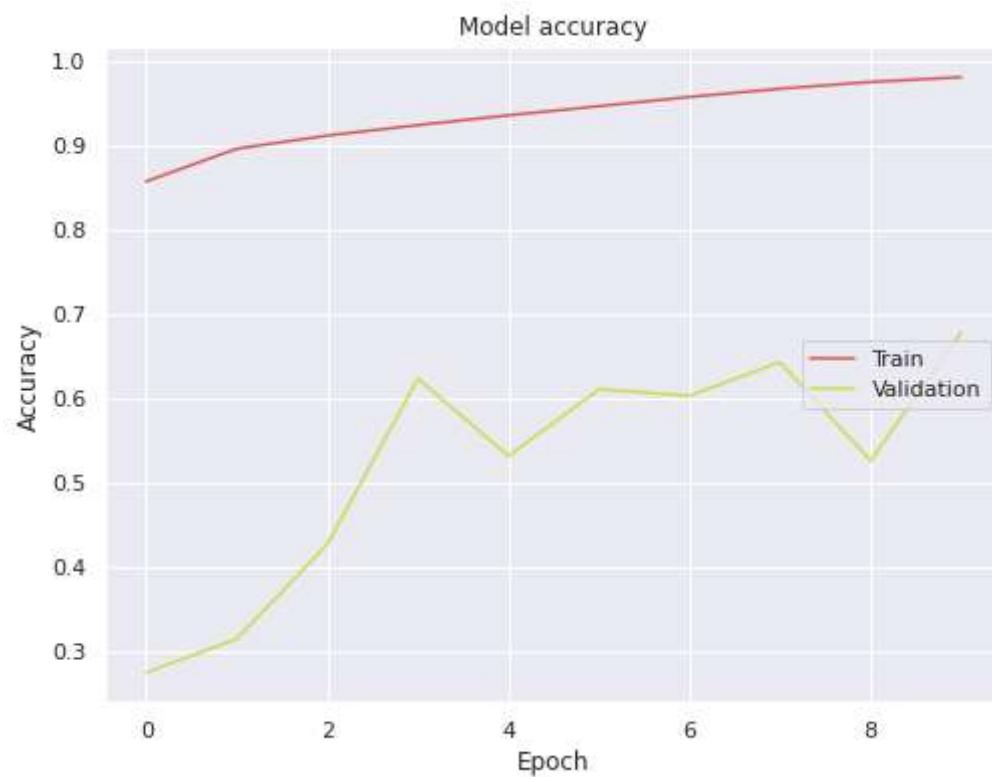
```
EPOCHS_N = 10
```

In [0]:

```
history = model.fit(x = x, y = y, epochs = EPOCHS_N,
                      validation_split = VAL_SPLIT_RATE, verbose = 0)
```

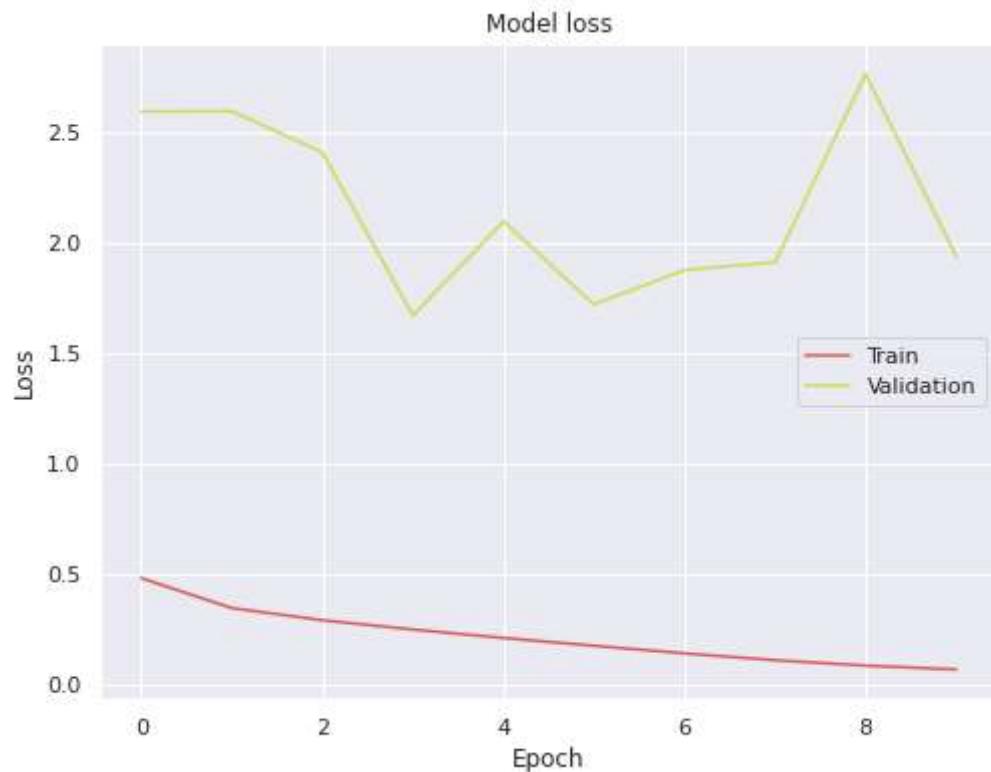
In [25]:

```
plot_accuracy(history, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [26]:

```
plot_loss(history)
```



In [27]:

```
results = model.evaluate(x_test, y_test)  
print('Test loss, test accuracy:', results)
```

Test loss, test accuracy: [0.47965525410249255, 0.9042104]

Точность построенной модели на тестовой выборке составила 90%.

## 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]:

```
from tensorflow.keras.layers import MaxPooling2D

model_2 = tf.keras.Sequential()

model_2.add(Conv2D(16, 3, padding = 'same', activation = 'relu',
                  input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)))
model_2.add(MaxPooling2D())
model_2.add(Flatten())
model_2.add(Dense(DENSE_LAYER_WIDTH, activation = 'relu'))
model_2.add(Dense(CLASSES_N))
```

In [0]:

```
model_2.compile(optimizer = 'sgd',
                 loss = cat_crossentropy,
                 metrics = ['categorical_accuracy'])
```

In [30]:

```
model_2.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 28, 28, 16)	160
max_pooling2d (MaxPooling2D)	(None, 14, 14, 16)	0
flatten_1 (Flatten)	(None, 3136)	0
dense_2 (Dense)	(None, 5000)	15685000
dense_3 (Dense)	(None, 10)	50010

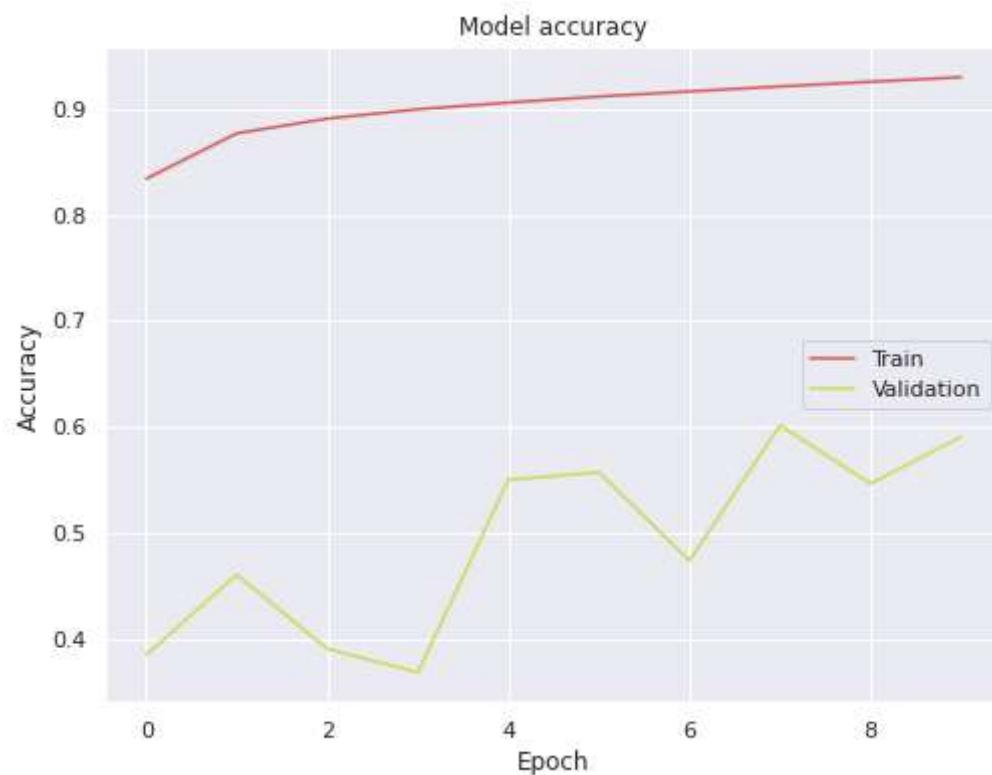
Total params: 15,735,170  
Trainable params: 15,735,170  
Non-trainable params: 0

In [0]:

```
history_2 = model_2.fit(x = x, y = y, epochs = EPOCHS_N,
                         validation_split = VAL_SPLIT_RATE, verbose = 0)
```

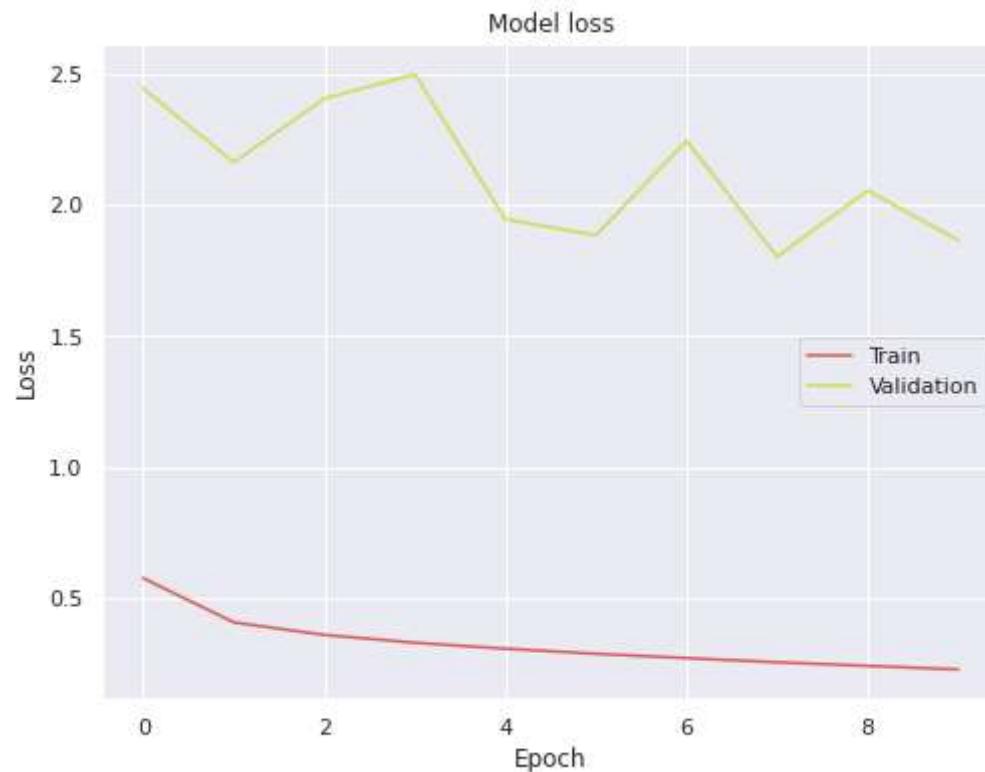
In [32]:

```
plot_accuracy(history_2, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [33]:

```
plot_loss(history_2)
```



In [34]:

```
results_2 = model_2.evaluate(x_test, y_test)  
print('Test loss, test accuracy:', results_2)
```

Test loss, test accuracy: [0.44243563031696087, 0.88264966]

Замена свёрточного слоя на операцию пулинга немножко снизила точность на тестовой выборке — до 88%.

## Задание 3

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

In [0]:

```
from tensorflow.keras.layers import AveragePooling2D

model_3 = tf.keras.Sequential()

model_3.add(Conv2D(6, kernel_size = (5, 5), strides = (1, 1),
                  activation = 'tanh', padding = 'same',
                  input_shape = (IMAGE_DIM_0, IMAGE_DIM_1, 1)))
model_3.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2),
                            padding = 'valid'))
model_3.add(Conv2D(16, kernel_size = (5, 5), strides = (1, 1),
                   activation = 'tanh', padding = 'valid'))
model_3.add(AveragePooling2D(pool_size = (2, 2), strides = (2, 2),
                            padding = 'valid'))
model_3.add(Flatten())
model_3.add(Dense(120, activation = 'tanh'))
model_3.add(Dense(84, activation = 'tanh'))
model_3.add(Dense(CLASSES_N, activation = 'softmax'))
```

In [0]:

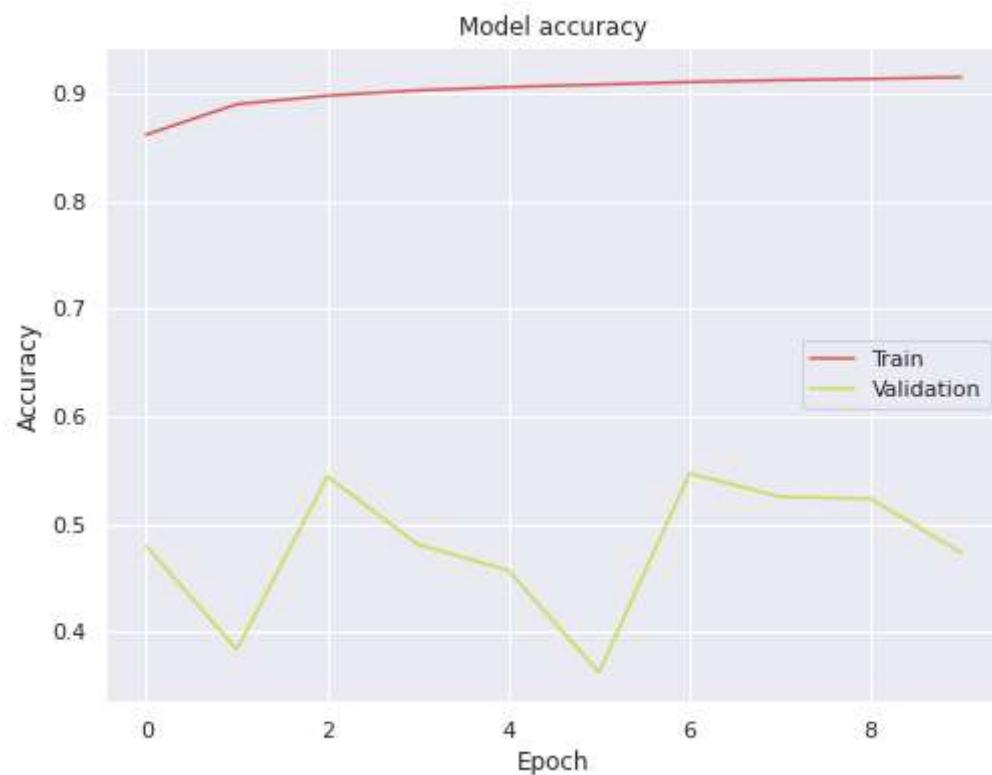
```
model_3.compile(optimizer = 'adam',
                 loss = 'categorical_crossentropy',
                 metrics = ['categorical_accuracy'])
```

In [0]:

```
history_3 = model_3.fit(x = x, y = y, epochs = EPOCHS_N,
                         validation_split = VAL_SPLIT_RATE, verbose = 0)
```

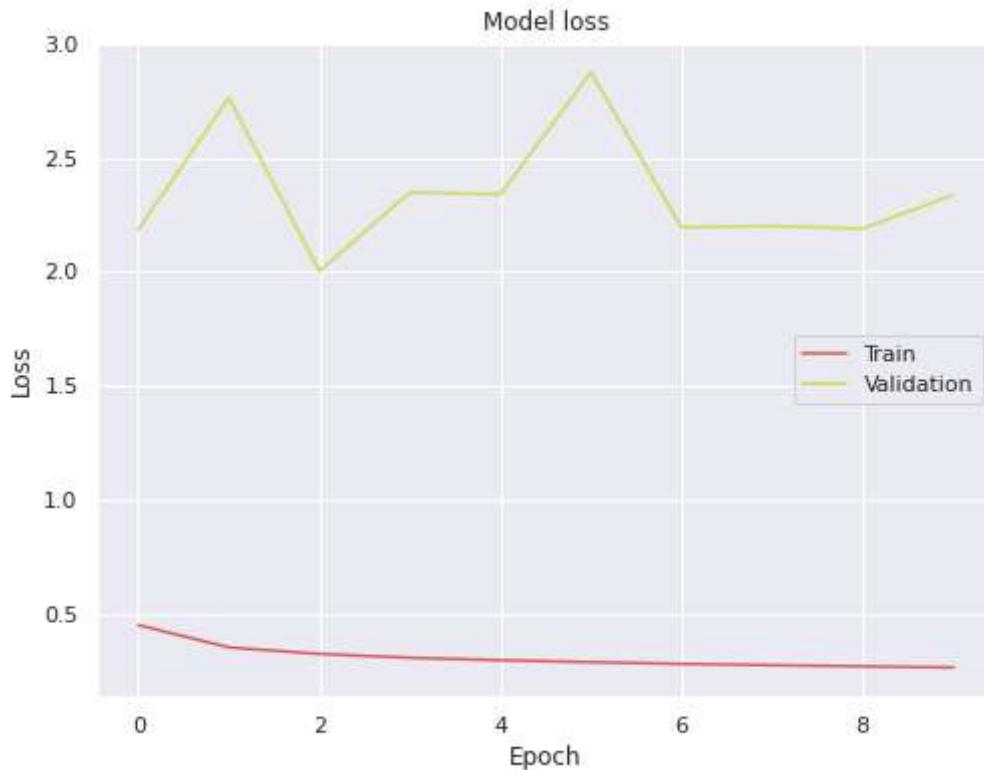
In [38]:

```
plot_accuracy(history_3, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [39]:

```
plot_loss(history_3)
```



In [40]:

```
results_3 = model_3.evaluate(x_test, y_test)
```

```
print('Test loss, test accuracy:', results_3)
```

```
Test loss, test accuracy: [0.49397167890073673, 0.8667388]
```

Удивительно, но *LeNet-5* показала результат хуже, чем первая и вторая — 86% на тестовой выборке. Объяснить это можно различиями в размерностях слоёв.

## Задание 4

Сравните максимальные точности моделей, построенных в лабораторных работах 1-3. Как можно объяснить полученные различия?

Результаты на валидационной выборке:

- логистическая регрессия — 81%;

- модель с только полносвязными слоями — 10%;
  - с регуляризацией и сбросом нейронов — 63%;
    - с адаптивным шагом — 64%;
- модель с двумя свёрточными слоями и одним полносвязным — 90%;
- модель с одним свёрточным слоем, операцией пулинга и одним полносвязным — 88%;
- *LeNet-5* — два свёрточных слоя, две операции пулинга, два полносвязных слоя — 86%.

Объяснение превосходства свёрточных сетей над полносвязными — такая архитектура лучше сочетается с природой изображений.

# Лабораторная работа №4

## Реализация приложения по распознаванию номеров домов

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

- 73257 изображений цифр в обучающей выборке;
- 26032 изображения цифр в тестовой выборке;
- 531131 изображения, которые можно использовать как дополнение к обучающей выборке;
- В двух форматах:
  - Оригинальные изображения с выделенными цифрами;
  - Изображения размером 32×32, содержащие одну цифру;
- Данные первого формата можно скачать по ссылкам:
  - <http://ufldl.stanford.edu/housenumbers/train.tar.gz> (<http://ufldl.stanford.edu/housenumbers/train.tar.gz>) (обучающая выборка);
  - <http://ufldl.stanford.edu/housenumbers/test.tar.gz> (<http://ufldl.stanford.edu/housenumbers/test.tar.gz>) (тестовая выборка);
  - <http://ufldl.stanford.edu/housenumbers/extrtar.gz> (<http://ufldl.stanford.edu/housenumbers/extrtar.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) ([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) ([http://ufldl.stanford.edu/housenumbers/test\\_32x32.mat](http://ufldl.stanford.edu/housenumbers/test_32x32.mat)) (тестовая выборка);
  - [http://ufldl.stanford.edu/housenumbers/extr\\_32x32.mat](http://ufldl.stanford.edu/housenumbers/extr_32x32.mat) ([http://ufldl.stanford.edu/housenumbers/extr\\_32x32.mat](http://ufldl.stanford.edu/housenumbers/extr_32x32.mat)) (дополнительные данные);
- Описание данных на английском языке доступно по ссылке:
  - <http://ufldl.stanford.edu/housenumbers/> (<http://ufldl.stanford.edu/housenumbers/>)

### Задание 1

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

Ознакомьтесь с имеющимися работами по данной тематике: англоязычная статья (<http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf> (<http://static.googleusercontent.com/media/research.google.com/en//pubs/archive/42241.pdf>)), видео на *YouTube* ([https://www.youtube.com/watch?v=vGPI\\_JvLoN0](https://www.youtube.com/watch?v=vGPI_JvLoN0)) ([https://www.youtube.com/watch?v=vGPI\\_JvLoN0](https://www.youtube.com/watch?v=vGPI_JvLoN0)).

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

In [0]:

```
import warnings  
  
warnings.filterwarnings('ignore')
```

In [0]:

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

In [0]:

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

In [0]:

```
import numpy as np
```

In [0]:

```
from tensorflow.keras.datasets import mnist  
  
(x_train, y_train), (x_test, y_test) = mnist.load_data()
```

In [0]:

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

In [0]:

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

In [8]:

```
from tensorflow.keras.utils import to_categorical  
  
y_train, y_test = to_categorical(y_train), to_categorical(y_test)  
  
y_train.shape
```

Out[8]:

```
(60000, 10)
```

In [0]:

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

In [0]:

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

In [11]:

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

Out[11]:

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

In [0]:

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

model = tf.keras.Sequential()

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

In [0]:

```
# 'sparse_categorical_crossentropy' gave NAN Loss

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

In [14]:

```
model.summary()
```

Model: "sequential"

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

In [0]:

```
EPOCHS_N = 20
```

In [0]:

```
history = model.fit(x = x_train, y = y_train, validation_split = 0.15,
                      epochs = EPOCHS_N, verbose = 0)
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_accuracy(_history,
                  _train_acc_name = 'accuracy',
                  _val_acc_name = 'val_accuracy'):

    plt.plot(_history.history[_train_acc_name])
    plt.plot(_history.history[_val_acc_name])

    plt.title('Model accuracy')

    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

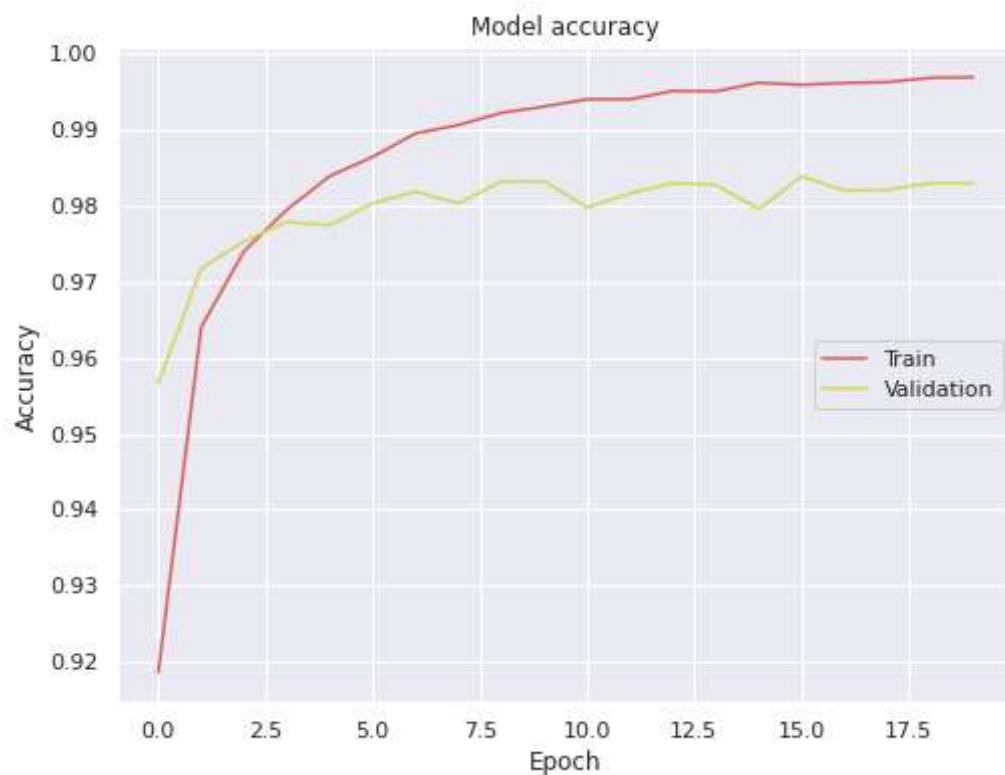
    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

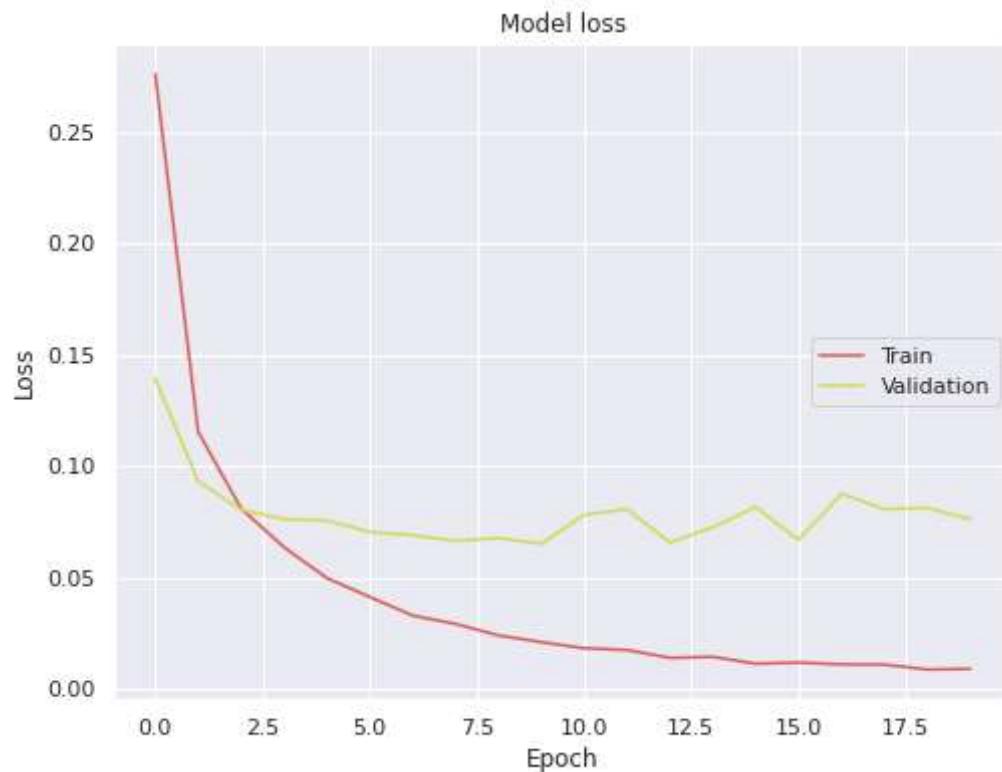
In [18]:

```
plot_accuracy(history, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [19]:

```
plot_loss(history)
```



In [20]:

```
results = model.evaluate(x_test, y_test)  
print('Test loss, test accuracy:', results)
```

```
313/313 [=====] - 1s 2ms/step - loss: 0.0643 - categorical_accuracy: 0.9835  
Test loss, test accuracy: [0.06433768570423126, 0.9835000038146973]
```

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

## Задание 2

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

## Одна цифра

In [0]:

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

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

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

In [0]:

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

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

    file_url_ = _url_folder + _name + _ext

    local_file_name_ = _name + '_' + _key + _local_ext

    urlretrieve(file_url_, local_file_name_)

    return local_file_name_

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

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

    dir_name_ = _name + '_' + _key

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

    os.remove(local_file_name_)

    return dir_name_
```

In [0]:

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

In [0]:

```
from scipy import io

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

In [25]:

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

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

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

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

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

In [0]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

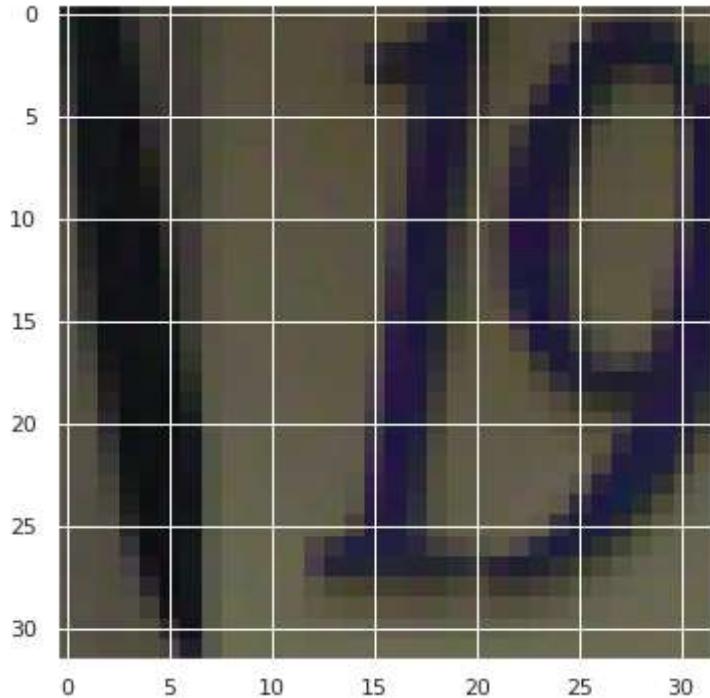
sns.set()

sns.set_palette(sns.color_palette('hls'))
```

In [28]:

```
plt.imshow(X_second_ds_train[0])
```

```
plt.show()
```



In [0]:

```
IMAGE_DIM_0_2 = X_second_ds_train.shape[-3]
IMAGE_DIM_1_2 = X_second_ds_train.shape[-2]
IMAGE_DIM_2_2 = X_second_ds_train.shape[-1]
```

In [0]:

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

In [0]:

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

In [0]:

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

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

In [0]:

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

In [34]:

```
model_2.summary()
```

Model: "sequential\_1"

Layer (type)	Output Shape	Param #
conv2d_2 (Conv2D)	(None, 32, 32, 6)	456
average_pooling2d_2 (Average)	(None, 16, 16, 6)	0
conv2d_3 (Conv2D)	(None, 12, 12, 16)	2416
average_pooling2d_3 (Average)	(None, 6, 6, 16)	0
flatten_1 (Flatten)	(None, 576)	0
dense_3 (Dense)	(None, 120)	69240
dense_4 (Dense)	(None, 84)	10164
dense_5 (Dense)	(None, 11)	935

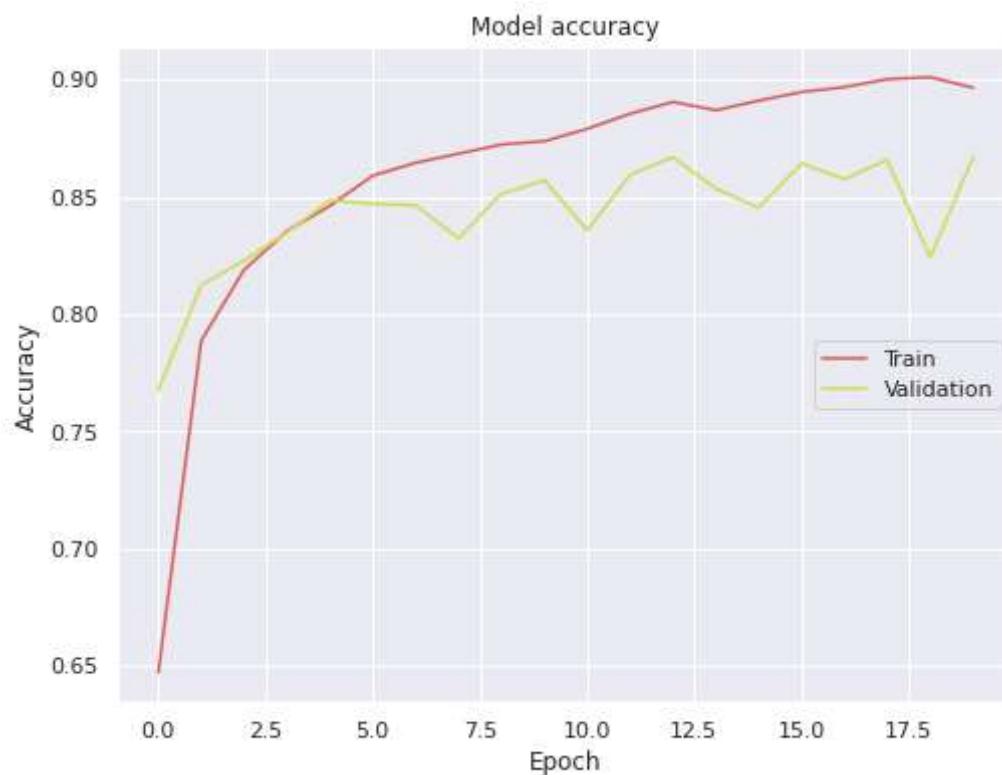
Total params: 83,211  
Trainable params: 83,211  
Non-trainable params: 0

In [0]:

```
history_2 = model_2.fit(x = X_second_ds_train, y = y_second_ds_train_cat,
                         validation_split = 0.15, epochs = EPOCHS_N, verbose = 0)
```

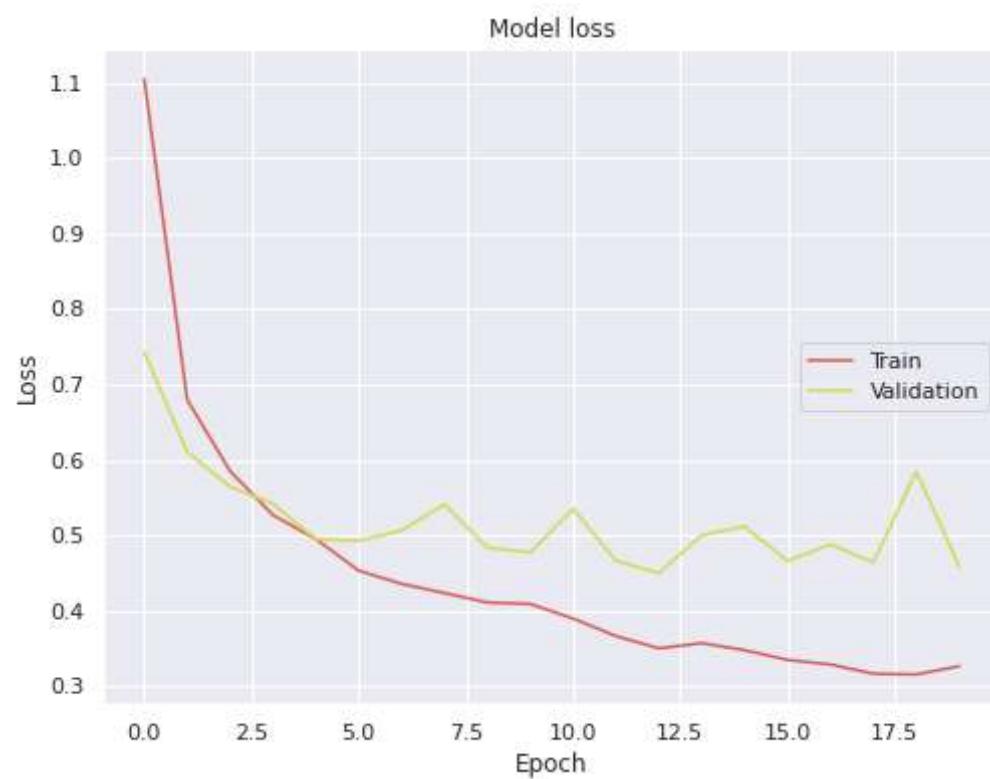
In [36]:

```
plot_accuracy(history_2, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [37]:

```
plot_loss(history_2)
```



In [38]:

```
results = model_2.evaluate(X_second_ds_test, y_second_ds_test_cat)
print('Test loss, test accuracy:', results)
```

```
814/814 [=====] - 2s 2ms/step - loss: 0.5257 - categorical_accuracy: 0.8460
Test loss, test accuracy: [0.5257437229156494, 0.8459588289260864]
```

Здесь в модели изменилось то, что добавился ещё один класс — *нет цифры*.

Эти данные более сложны для распознавания, что повлияло на результат — точность распознавания на тестовой выборке составила 84%.

## Несколько цифр

Загрузим первый датасет — реальные изображения с несколькими цифрами и рамками границ.

In [0]:

```
from imageio import imread
import pandas as pd

def image_to_array(_image):
    try:
        array_ = imread(_image)

        return True, array_
    except:
        return False, None

def dir_to_dataframe(_dir_path):
    data_ = []
    files_ = sorted(os.listdir(_dir_path))

    for f in files_:
        file_path_ = os.path.join(_dir_path, f)

        can_read_, im = image_to_array(file_path_)

        if can_read_:
            data_.append(im)

    dataframe_ = pd.DataFrame()
    dataframe_[ 'data' ] = np.array(data_)

    return dataframe_
```

In [0]:

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

In [0]:

```
first_ds_train_subdir = os.path.join(first_ds_train_dir, 'train')  
first_ds_test_subdir = os.path.join(first_ds_test_dir, 'test')
```

In [0]:

```
first_ds_train_images_df = dir_to_dataframe(first_ds_train_subdir)  
first_ds_test_images_df = dir_to_dataframe(first_ds_test_subdir)
```

In [0]:

```
import h5py  
  
first_ds_train_boxes_mat = h5py.File(  
    os.path.join(first_ds_train_subdir, 'digitStruct.mat'), 'r')  
first_ds_test_boxes_mat = h5py.File(  
    os.path.join(first_ds_test_subdir, 'digitStruct.mat'), 'r')
```

In [0]:

```
import numpy as np
import pickle
import h5py

def mat_to_pickle(_mat_path, _key):

    f = h5py.File(_mat_path, 'r')

    metadata = {}

    metadata['height'] = []
    metadata['label'] = []
    metadata['left'] = []
    metadata['top'] = []
    metadata['width'] = []

    def print_attrs(name, obj):
        vals = []
        if obj.shape[0] == 1:
            vals.append(int(obj[0][0]))
        else:
            for k in range(obj.shape[0]):
                vals.append(int(f[obj[k][0]][0][0]))
        metadata[name].append(vals)

    for item in f['/digitStruct/bbox']:
        f[item[0]].visititems(print_attrs)

    with open('{}.pickle'.format(_key), 'wb') as pf:
        pickle.dump(metadata, pf, pickle.HIGHEST_PROTOCOL)
```

In [0]:

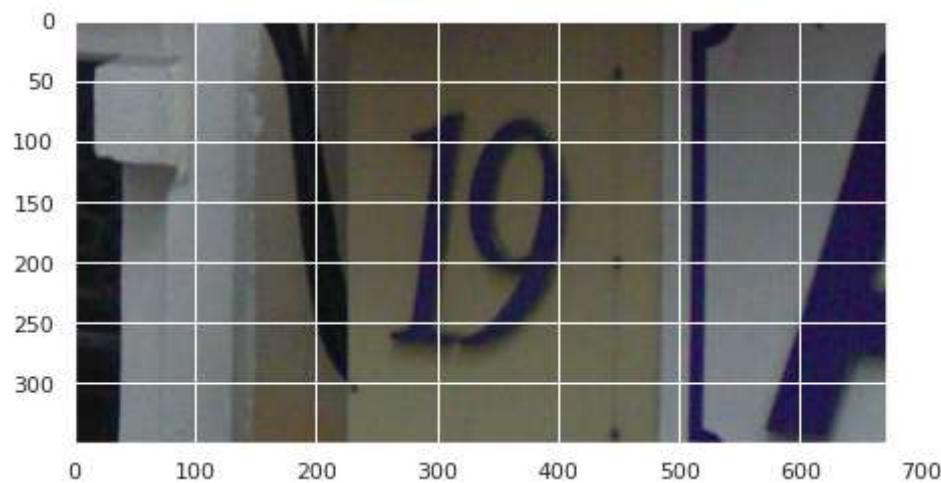
```
mat_to_pickle(
    os.path.join(first_ds_train_subdir, 'digitStruct.mat'), 'train_bbox')
mat_to_pickle(
    os.path.join(first_ds_test_subdir, 'digitStruct.mat'), 'test_bbox')
```

In [0]:

```
train_bbox_data = np.load('train_bbox.pickle', allow_pickle = True)
test_bbox_data = np.load('test_bbox.pickle', allow_pickle = True)
```

In [47]:

```
plt.imshow(first_ds_train_images_df['data'][0])  
plt.show()
```



In [48]:

```
train_bbox_data['height'][5]
```

Out[48]:

```
[21, 21]
```

In [0]:

```
MAX_DIGITS = 6
```

In [0]:

```
def to_full_df(_ds_images_df, _bbox_data):  
    LENGTH = len(_bbox_data['height'])  
  
    BBOX_SHAPE_TUPLE = (LENGTH, MAX_DIGITS)  
  
    bbox_heights = np.zeros(BBOX_SHAPE_TUPLE)  
    bbox_labels = np.zeros(BBOX_SHAPE_TUPLE)  
    bbox_lefts = np.zeros(BBOX_SHAPE_TUPLE)  
    bbox_tops = np.zeros(BBOX_SHAPE_TUPLE)  
    bbox_widths = np.zeros(BBOX_SHAPE_TUPLE)  
  
    for i in range(LENGTH):  
        j = 0  
  
        l = len(_bbox_data['height'][i])  
  
        while j < l:  
            bbox_heights[i][j] = _bbox_data['height'][i][j]  
            bbox_labels[i][j] = _bbox_data['label'][i][j]  
            bbox_lefts[i][j] = _bbox_data['left'][i][j]  
            bbox_tops[i][j] = _bbox_data['top'][i][j]  
            bbox_widths[i][j] = _bbox_data['width'][i][j]  
  
            j = j + 1  
  
    data_dict_ = {  
        'data': _ds_images_df['data'],  
  
        'height_0': bbox_heights[:, 0],  
        'label_0': bbox_labels[:, 0],  
        'left_0': bbox_lefts[:, 0],  
        'top_0': bbox_tops[:, 0],  
        'width_0': bbox_widths[:, 0],  
  
        'height_1': bbox_heights[:, 1],  
        'label_1': bbox_labels[:, 1],  
        'left_1': bbox_lefts[:, 1],  
        'top_1': bbox_tops[:, 1],  
        'width_1': bbox_widths[:, 1],  
  
        'height_2': bbox_heights[:, 2],  
        'label_2': bbox_labels[:, 2],  
        'left_2': bbox_lefts[:, 2],  
        'top_2': bbox_tops[:, 2],  
        'width_2': bbox_widths[:, 2],  
  
        'height_3': bbox_heights[:, 3],  
        'label_3': bbox_labels[:, 3],  
        'left_3': bbox_lefts[:, 3],  
        'top_3': bbox_tops[:, 3],  
        'width_3': bbox_widths[:, 3],  
  
        'height_4': bbox_heights[:, 4],  
        'label_4': bbox_labels[:, 4],  
        'left_4': bbox_lefts[:, 4],  
        'top_4': bbox_tops[:, 4],
```

```

'width_4': bbox_widths[:, 4],  

'height_5': bbox_heights[:, 5],  

'label_5': bbox_labels[:, 5],  

'left_5': bbox_lefts[:, 5],  

'top_5': bbox_tops[:, 5],  

'width_5': bbox_widths[:, 5],  

}  
  

full_ds_ = pd.DataFrame(data_dict_,  

                         columns = [  

                           'data',  

                           'height_0',  

                           'label_0',  

                           'left_0',  

                           'top_0',  

                           'width_0',  

                           'height_1',  

                           'label_1',  

                           'left_1',  

                           'top_1',  

                           'width_1',  

                           'height_2',  

                           'label_2',  

                           'left_2',  

                           'top_2',  

                           'width_2',  

                           'height_3',  

                           'label_3',  

                           'left_3',  

                           'top_3',  

                           'width_3',  

                           'height_4',  

                           'label_4',  

                           'left_4',  

                           'top_4',  

                           'width_4',  

                           'height_5',  

                           'label_5',  

                           'left_5',  

                           'top_5',  

                           'width_5',  

                         ])  
  

return full_ds_

```

In [0]:

```

first_ds_train_full_df = to_full_df(first_ds_train_images_df, train_bbox_data)
first_ds_test_full_df = to_full_df(first_ds_test_images_df, test_bbox_data)

```

In [0]:

```
def no_more_than_two_digits(_full_df):  
  
    _2_digits_df = _full_df[_full_df['height_2'] == 0.0].reset_index()  
  
    _2_digits_df = _2_digits_df.drop(columns = [  
        'height_2',  
        'label_2',  
        'left_2',  
        'top_2',  
        'width_2',  
  
        'height_3',  
        'label_3',  
        'left_3',  
        'top_3',  
        'width_3',  
  
        'height_4',  
        'label_4',  
        'left_4',  
        'top_4',  
        'width_4',  
  
        'height_5',  
        'label_5',  
        'left_5',  
        'top_5',  
        'width_5'  
    ])  
  
    return _2_digits_df
```

In [0]:

```
first_ds_train_2_digits_df = no_more_than_two_digits(first_ds_train_full_df)  
first_ds_test_2_digits_df = no_more_than_two_digits(first_ds_test_full_df)
```

In [0]:

```
from math import ceil

def get_image_central_square(_image):

    dim_0 = _image.shape[0]
    dim_1 = _image.shape[1]

    if dim_0 == 0 or dim_1 == 0:

        print(_image.shape)

    cutoff_ = ceil(abs(dim_0 - dim_1) / 2)

    if dim_0 > dim_1:
        cut_image_ = _image[cutoff_: -cutoff_,
                            :, :]
    elif dim_0 < dim_1:
        cut_image_ = _image[:, cutoff_: -cutoff_,
                            :]
    else:
        cut_image_ = _image[:, :, :]

    return cut_image_
```

In [0]:

```
NEW_IMAGE_DIM = 100
```

In [0]:

```
import cv2

def resize_image(_image, _dim_0 = NEW_IMAGE_DIM, _dim_1 = NEW_IMAGE_DIM):

    try:
        resized_ = cv2.resize(_image, dsize = (_dim_0, _dim_1),
                             interpolation = cv2.INTER_CUBIC)
    except:
        print(_image.shape)

    return resized_
```

In [0]:

```
def process_image(_image):

    squared_ = get_image_central_square(_image)

    resized_ = resize_image(squared_)

    return resized_
```

In [0]:

```
def get_digits_n_from_row(_row):  
  
    if _row['height_1'] != 0.0:  
        return 2  
  
    if _row['height_0'] != 0.0:  
        return 1  
  
    return 0
```

In [0]:

```
def to_new_format_dataframe(_dataframe):  
  
    df_copy_ = _dataframe.copy()  
  
    rrrr = df_copy_.apply(lambda row: process_image(row['data']), axis = 1)  
  
    df_copy_.drop(columns = ['data'])  
  
    df_copy_['data'] = rrrr  
  
    nnnn = df_copy_.apply(lambda row: get_digits_n_from_row(row), axis = 1)  
  
    df_copy_['digits_n'] = nnnn  
  
    df_copy_['digit_0'] = df_copy_['label_0'].astype(int)  
    df_copy_['digit_1'] = df_copy_['label_1'].astype(int)  
  
    df_copy_ = df_copy_.drop(columns = [  
                                         'height_0',  
                                         'label_0',  
                                         'left_0',  
                                         'top_0',  
                                         'width_0',  
  
                                         'height_1',  
                                         'label_1',  
                                         'left_1',  
                                         'top_1',  
                                         'width_1'  
                                         ])  
  
    return df_copy_
```

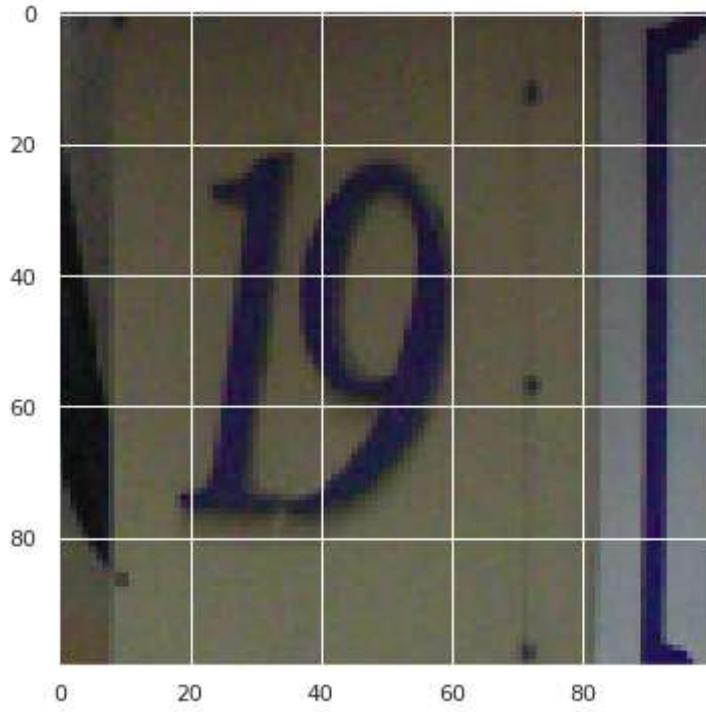
In [0]:

```
train_resized_df = to_new_format_dataframe(first_ds_train_2_digits_df)  
test_resized_df = to_new_format_dataframe(first_ds_test_2_digits_df)
```

In [61]:

```
plt.imshow(train_resized_df[ 'data' ][0])
```

```
plt.show()
```



In [62]:

```
train_resized_df[ 'digits_n' ][0], train_resized_df[ 'digit_0' ][0], train_resized_df[ 'digit_1' ]
```

Out[62]:

```
(2, 1, 9)
```

### Задание 3

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

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

### Задание 4

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

# Лабораторная работа №5

## Применение сверточных нейронных сетей (бинарная классификация)

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

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

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

### Задание 1

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

In [0]:

```
import warnings  
  
warnings.filterwarnings('ignore')
```

In [2]:

```
from google.colab import drive  
  
drive.mount('/content/drive', force_remount = True)
```

Mounted at /content/drive

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2/dogs-vs-cats'  
  
import sys  
  
sys.path.append(BASE_DIR)  
  
import os
```

In [0]:

```
TRAIN_ARCHIVE_NAME = 'train.zip'  
TEST_ARCHIVE_NAME = 'test1.zip'  
  
LOCAL_DIR_NAME = 'dogs-vs-cats'
```

In [0]:

```
from zipfile import ZipFile

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

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

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_accuracy(_history,
                  _train_acc_name = 'accuracy',
                  _val_acc_name = 'val_accuracy'):

    plt.plot(_history.history[_train_acc_name])
    plt.plot(_history.history[_val_acc_name])

    plt.title('Model accuracy')

    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

In [7]:

```
from matplotlib.image import imread

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

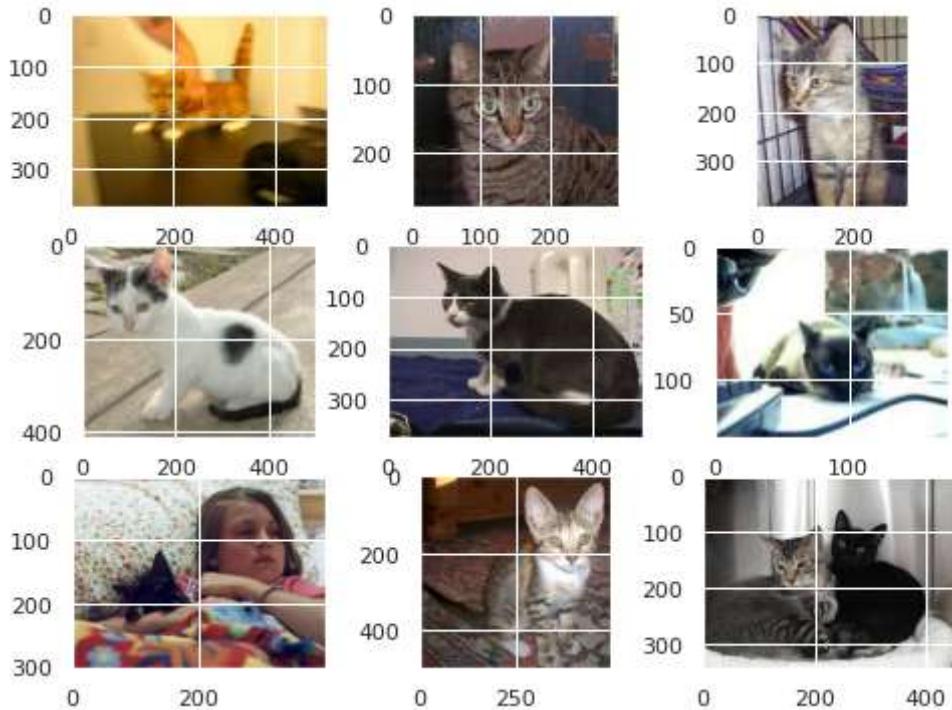
for i in range(9):

    plt.subplot(330 + 1 + i)

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

    plt.imshow(image_)

plt.show()
```



Изображения необходимо привести к одному размеру.

In [0]:

```
NEW_IMAGE_WIDTH = 100
```

In [9]:

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

def dir_to_dataset(_dir_path):

    photos_, labels_ = [], []
    for file_ in listdir(_dir_path):
        if file_.startswith('cat'):
            label_ = 1.0
        else:
            label_ = 0.0
        photo_ = load_img(join(_dir_path, file_), target_size = (NEW_IMAGE_WIDTH, NEW_IMAGE_WIDTH))
        photo_ = img_to_array(photo_)
        photos_.append(photo_)
        labels_.append(label_)

    photos_norm_ = tf.keras.utils.normalize(photos_, axis = 1)
    return asarray(photos_norm_), asarray(labels_)
```

Using TensorFlow backend.

In [0]:

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

In [0]:

```
import tensorflow as tf
```

In [0]:

```
import numpy as np
```

In [0]:

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

In [0]:

```
TEST_LEN_HALF = 1000
```

In [15]:

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

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

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

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

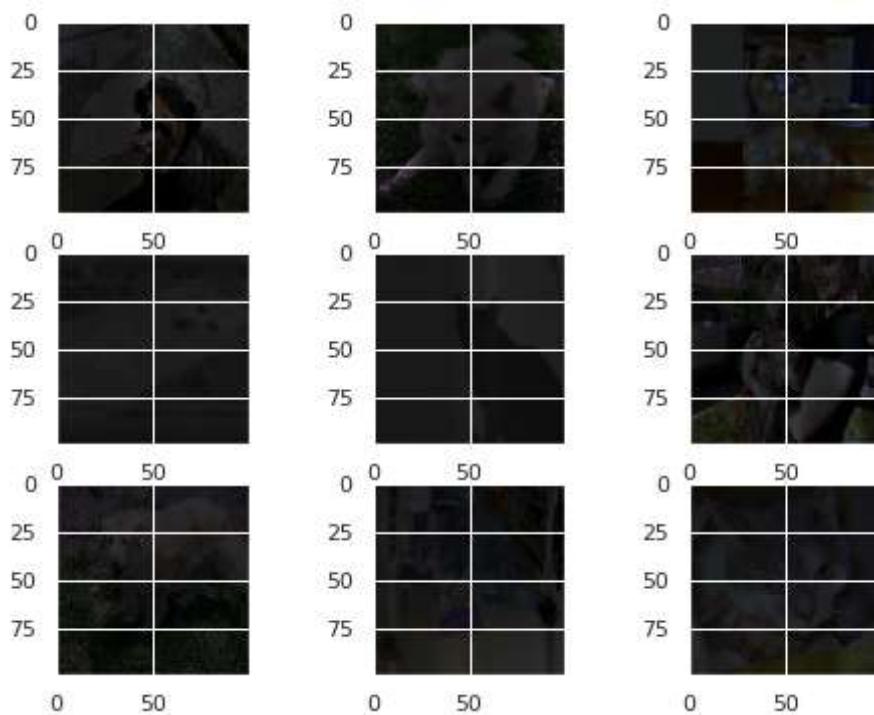
In [16]:

```
for i in range(9):

    plt.subplot(330 + 1 + i)

    plt.imshow(X[i])

plt.show()
```



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

## Задание 2

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

In [0]:

```
from tensorflow import keras
```

In [18]:

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

model = tf.keras.Sequential()

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

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

model.summary()
```

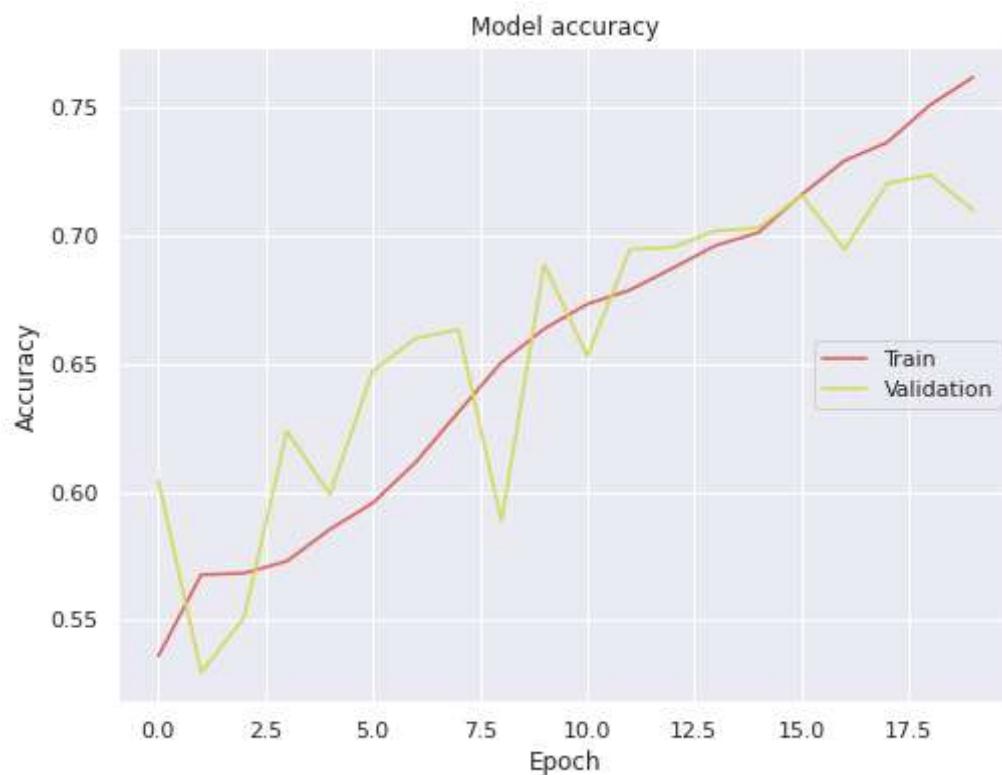
Model: "sequential"

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

In [0]:

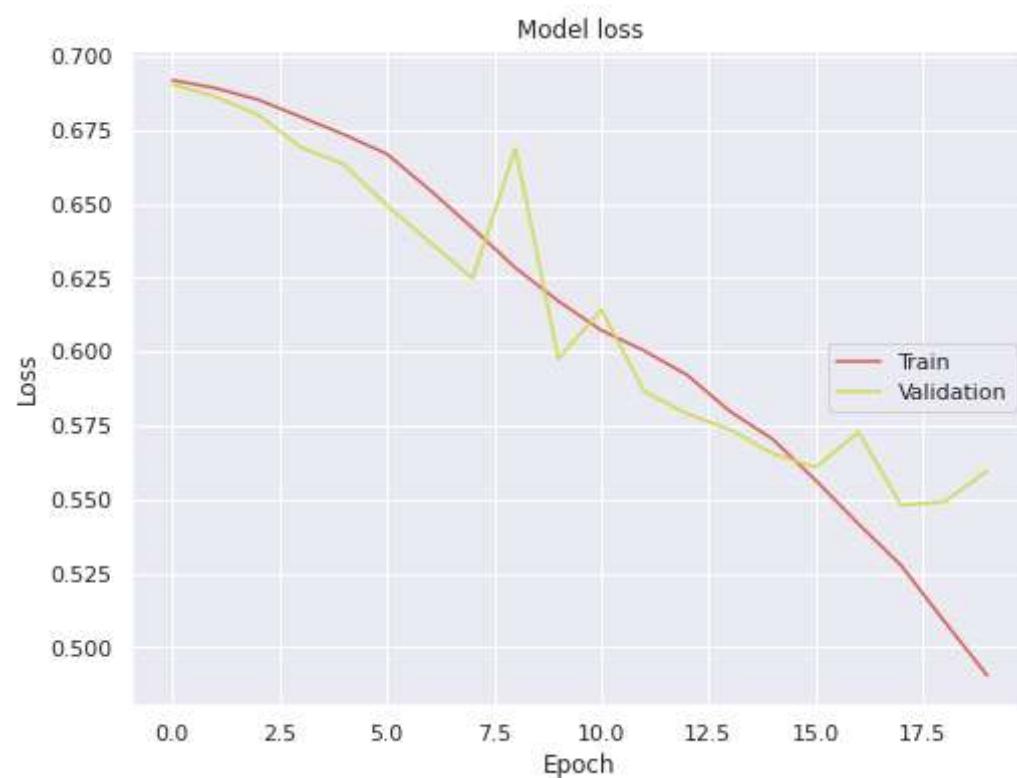
In [20]:

```
plot_accuracy(history)
```



In [21]:

```
plot_loss(history)
```



In [22]:

```
results = model.evaluate(X_test, y_test)
print('Test loss, test accuracy:', results)

63/63 [=====] - 0s 4ms/step - loss: 0.5706 - accuracy: 0.6935
Test loss, test accuracy: [0.5705881118774414, 0.6934999823570251]
```

Результат — 69% на тестовой выборке.

## Задание 3

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

In [0]:

```
def augment_image(image):
    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize_with_crop_or_pad(image,
                                             NEW_IMAGE_WIDTH + 40,
                                             NEW_IMAGE_WIDTH + 40)
    image = tf.image.random_crop(image,
                                 size = [NEW_IMAGE_WIDTH, NEW_IMAGE_WIDTH, 3])
    return image.numpy()
```

In [24]:

```
X_augmented = np.zeros_like(X)

for i, img in enumerate(X):
    X_augmented[i] = augment_image(img)

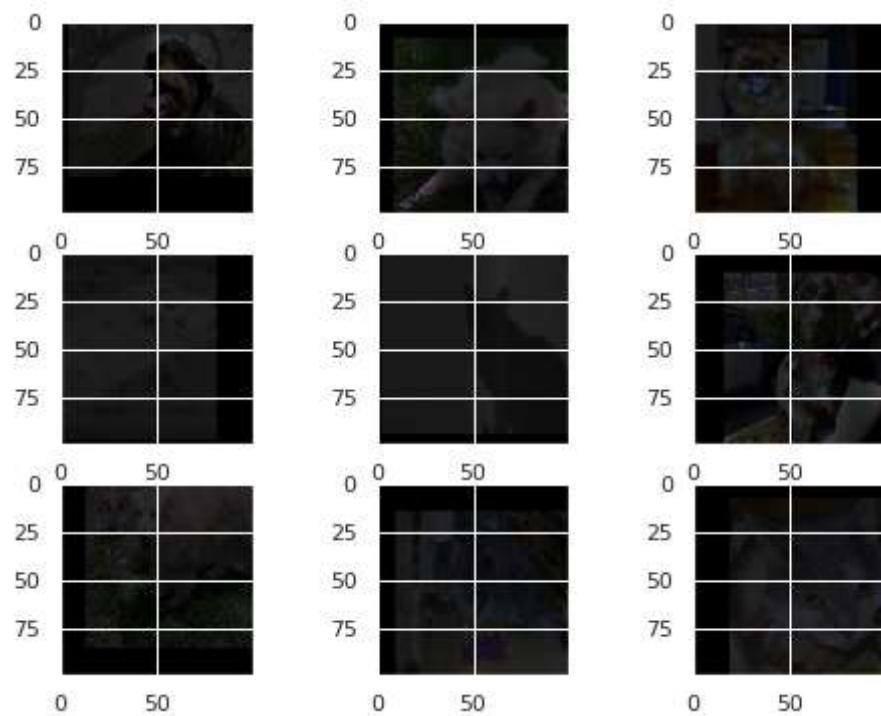
X_augmented.shape
```

Out[24]:

(23000, 100, 100, 3)

In [25]:

```
for i in range(9):  
    plt.subplot(330 + 1 + i)  
    plt.imshow(X_augmented[i])  
  
plt.show()
```



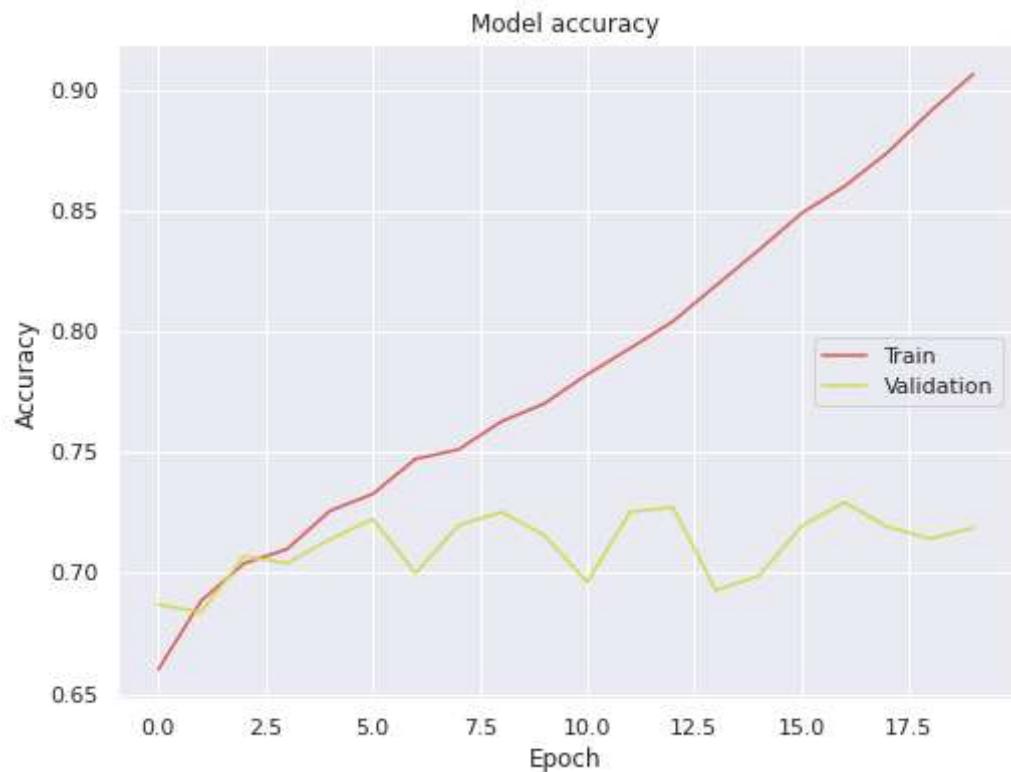
In [0]:

```
y_augmented = y
```

In [0]:

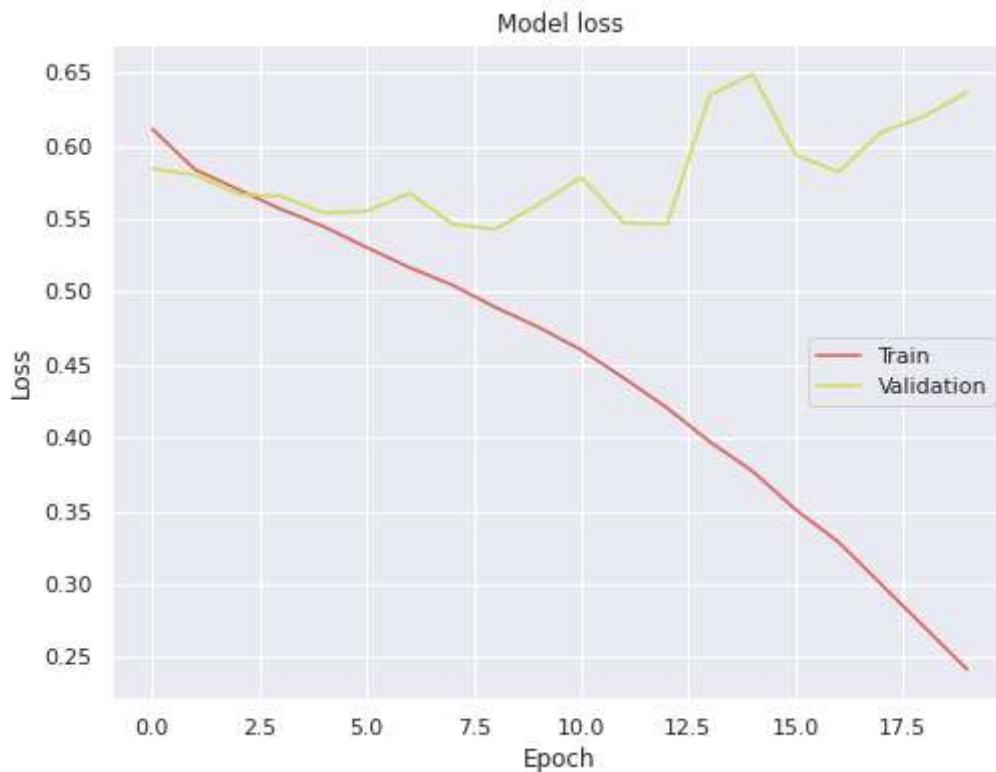
In [28]:

```
plot_accuracy(history_2)
```



In [29]:

```
plot_loss(history_2)
```



In [30]:

```
results_2 = model.evaluate(X_test, y_test)  
print('Test loss, test accuracy:', results_2)
```

```
63/63 [=====] - 0s 4ms/step - loss: 0.6174 - accuracy: 0.7590  
Test loss, test accuracy: [0.6173917651176453, 0.7590000033378601]
```

После того, как сеть обучилась на тех же данных, к которым был применён data augmentation, точность предсказания увеличилась — до 75%.

## 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?

# Лабораторная работа №6

## Применение сверточных нейронных сетей (многоклассовая классификация)

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 (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>)

### Задание 1

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

In [0]:

```
import warnings  
warnings.filterwarnings('ignore')
```

In [2]:

```
from google.colab import drive  
drive.mount('/content/drive', force_remount = True)
```

Mounted at /content/drive

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2'  
  
import sys  
  
sys.path.append(BASE_DIR)  
  
import os
```

In [0]:

```
DATA_ARCHIVE_NAME = 'sign-language-mnist.zip'  
  
LOCAL_DIR_NAME = 'sign-language'
```

In [0]:

```
from zipfile import ZipFile

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

In [0]:

```
TRAIN_FILE_PATH = 'sign-language/train/sign_mnist_train.csv'
TEST_FILE_PATH = 'sign-language/train/sign_mnist_test.csv'
```

In [0]:

```
import pandas as pd

train_df = pd.read_csv(TRAIN_FILE_PATH)
test_df = pd.read_csv(TEST_FILE_PATH)
```

In [8]:

```
train_df.shape, test_df.shape
```

Out[8]:

```
((27455, 785), (7172, 785))
```

In [0]:

```
IMAGE_DIM = 28
```

In [0]:

```
def row_to_label(_row):
    return _row[0]

def row_to_one_image(_row):
    return _row[1:].values.reshape((IMAGE_DIM, IMAGE_DIM, 1))
```

In [0]:

```
def to_images_and_labels(_dataframe):

    llll = _dataframe.apply(lambda row: row_to_label(row), axis = 1)
    mmmm = _dataframe.apply(lambda row: row_to_one_image(row), axis = 1)

    data_dict_ = { 'label': llll, 'image': mmmm }

    reshaped_ = pd.DataFrame(data_dict_, columns = ['label', 'image'])

    return reshaped_
```

In [0]:

```
train_df_reshaped = to_images_and_labels(train_df)
test_df_reshaped = to_images_and_labels(test_df)
```

## Задание 2

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

Возьмём *LeNet-5*.

In [0]:

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

In [0]:

```
import tensorflow as tf
```

In [0]:

```
from tensorflow.keras.utils import to_categorical
import numpy as np

X_train = tf.keras.utils.normalize(np.asarray(list(train_df_reshaped['image'])),
                                   axis = 1)
X_test = tf.keras.utils.normalize(np.asarray(list(test_df_reshaped['image'])),
                                   axis = 1)

y_train = to_categorical(train_df_reshaped['label']
                        .astype('category').cat.codes.astype('int32'))
y_test = to_categorical(test_df_reshaped['label']
                        .astype('category').cat.codes.astype('int32'))
```

In [16]:

```
X_train.shape, y_train.shape, X_test.shape, y_test.shape
```

Out[16]:

```
((27455, 28, 28, 1), (27455, 24), (7172, 28, 28, 1), (7172, 24))
```

In [0]:

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

In [0]:

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

model = tf.keras.Sequential()

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

In [0]:

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

In [20]:

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
=====		
conv2d (Conv2D)	(None, 28, 28, 6)	156
=====		
average_pooling2d (AveragePo	(None, 14, 14, 6)	0
=====		
conv2d_1 (Conv2D)	(None, 10, 10, 16)	2416
=====		
average_pooling2d_1 (Average	(None, 5, 5, 16)	0
=====		
flatten (Flatten)	(None, 400)	0
=====		
dense (Dense)	(None, 120)	48120
=====		
dense_1 (Dense)	(None, 84)	10164
=====		
dense_2 (Dense)	(None, 24)	2040
=====		
Total params: 62,896		
Trainable params: 62,896		
Non-trainable params: 0		

In [0]:

```
history = model.fit(x = X_train, y = y_train, epochs = 20,
                     validation_split = 0.15, verbose = 0)
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_accuracy(_history,
                  _train_acc_name = 'accuracy',
                  _val_acc_name = 'val_accuracy'):

    plt.plot(_history.history[_train_acc_name])
    plt.plot(_history.history[_val_acc_name])

    plt.title('Model accuracy')

    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

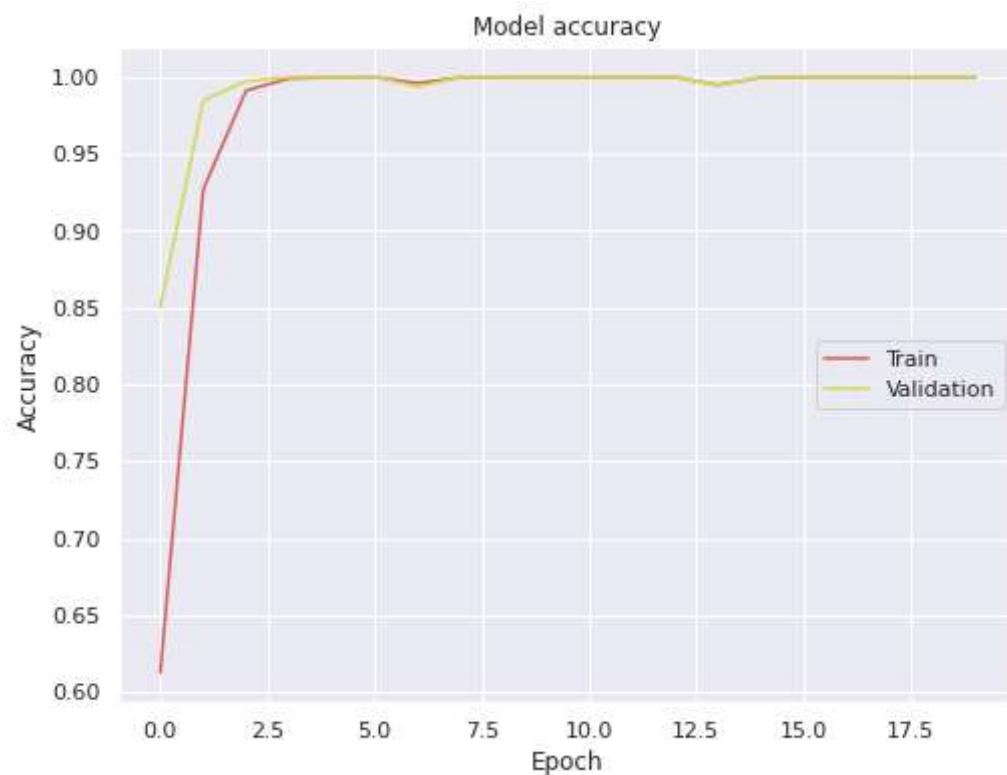
    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

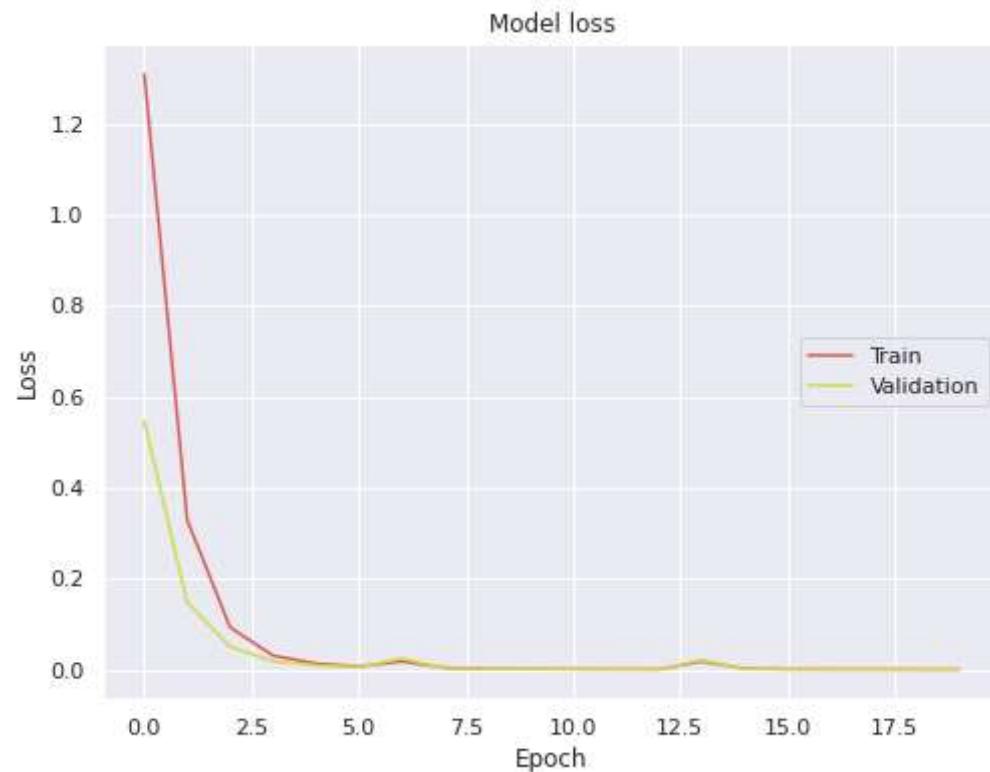
In [23]:

```
plot_accuracy(history, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [24]:

```
plot_loss(history)
```



In [25]:

```
results = model.evaluate(X_test, y_test)  
print('Test loss, test accuracy:', results)
```

```
225/225 [=====] - 1s 2ms/step - loss: 0.7977 - categorical_accuracy: 0.8358  
Test loss, test accuracy: [0.7976991534233093, 0.835750162601471]
```

За 20 эпох удалось достичь точности 83% на тестовой выборке.

### **Задание 3**

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

In [0]:

```
def augment_image(image):

    image = tf.image.convert_image_dtype(image, tf.float32)
    image = tf.image.resize_with_crop_or_pad(image, IMAGE_DIM + 6, IMAGE_DIM + 6)
    image = tf.image.random_crop(image, size = [IMAGE_DIM, IMAGE_DIM, 3])

return image.numpy()
```

In [27]:

```
X_train_augmented = np.zeros_like(X_train)
```

```
for i, img in enumerate(X_train):  
    X_train_augmented[i] = augment_image(img)  
  
X_train_augmented.shape
```

Out[27]:

(27455, 28, 28, 1)

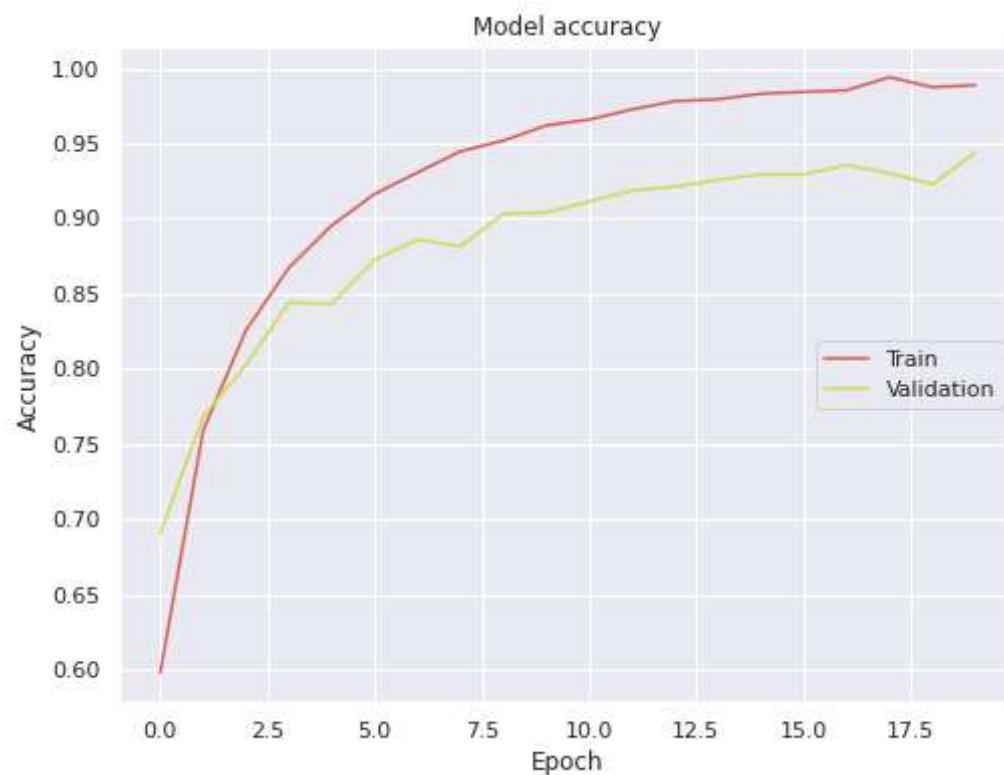
In [0]:

```
y_train_augmented = y_train
```

In [0]:

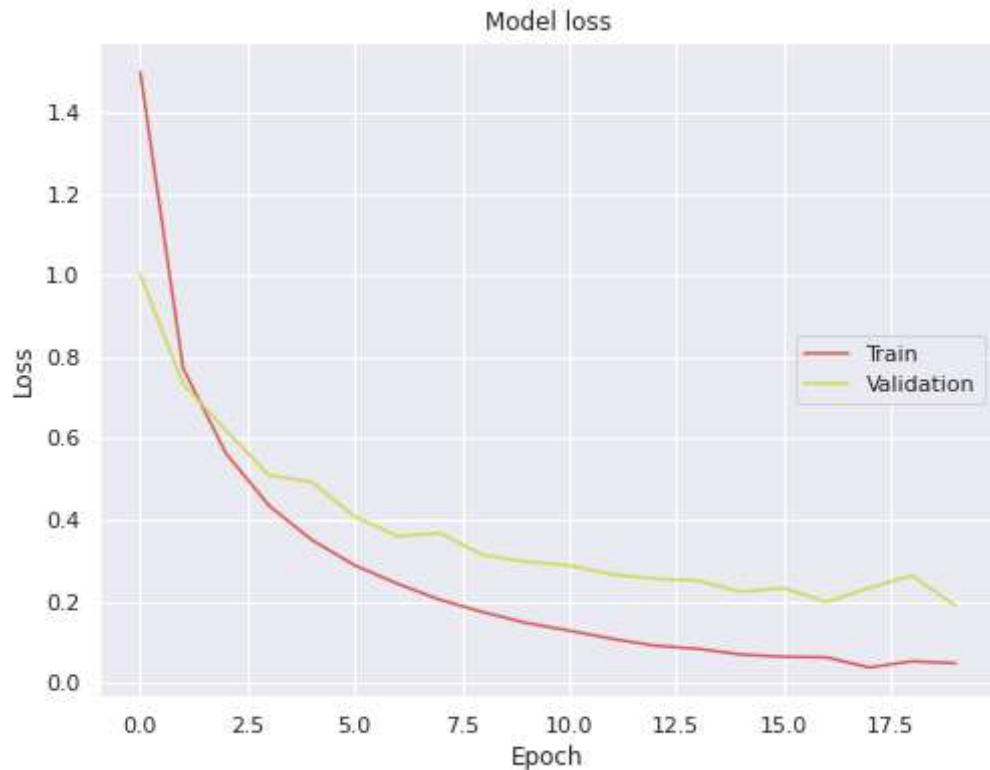
In [30]:

```
plot_accuracy(history_2, 'categorical_accuracy', 'val_categorical_accuracy')
```



In [31]:

```
plot_loss(history_2)
```



In [32]:

```
results_2 = model.evaluate(X_test, y_test)
```

```
print('Test loss, test accuracy:', results_2)
```

```
225/225 [=====] - 1s 2ms/step - loss: 0.3523 - categorical_accuracy: 0.9131
Test loss, test accuracy: [0.3523258864879608, 0.9131343960762024]
```

После того, как сеть обучилась на тех же данных, к которым был применён *data augmentation*, точность предсказания на тестовой выборке увеличилась до 91%.

## 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?

# Лабораторная работа №7

## Рекуррентные нейронные сети для анализа текста

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.

Данные можно скачать на сайте Kaggle: <https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset> (<https://www.kaggle.com/iarunava/imdb-movie-reviews-dataset>) <https://ai.stanford.edu/~amaas/data/sentiment/> (<https://ai.stanford.edu/~amaas/data/sentiment/>)

### Задание 1

Загрузите данные. Преобразуйте текстовые файлы во внутренние структуры данных, которые используют индексы вместо слов.

Будем брать первые MAX\_LENGTH слов, а если в отзыве слов меньше, чем это число, то применять паддинг.

In [0]:

```
import warnings  
  
warnings.filterwarnings('ignore')
```

In [0]:

```
from google.colab import drive  
  
drive.mount('/content/drive', force_remount = True)
```

Mounted at /content/drive

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2'  
  
import sys  
  
sys.path.append(BASE_DIR)  
  
import os
```

In [0]:

```
DATA_ARCHIVE_NAME = 'imdb-dataset-of-50k-movie-reviews.zip'
```

```
LOCAL_DIR_NAME = 'imdb-sentiments'
```

In [0]:

```
from zipfile import ZipFile

with ZipFile(os.path.join(BASE_DIR, DATA_ARCHIVE_NAME), 'r') as zip_:
    zip_.extractall(LOCAL_DIR_NAME)
```

In [0]:

```
DATA_FILE_PATH = 'imdb-sentiments/IMDB Dataset.csv'
```

In [0]:

```
import pandas as pd

all_df = pd.read_csv(DATA_FILE_PATH)
```

In [0]:

```
df_test = all_df.sample(frac = 0.1)

df_train = all_df.drop(df_test.index)
```

In [0]:

```
df_train.shape, df_test.shape
```

Out[9]:

```
((45000, 2), (5000, 2))
```

In [0]:

```
import nltk

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Package punkt is already up-to-date!
```

Out[10]:

```
True
```

In [0]:

```
MAX_LENGTH = 40
STRING_DTYPE = '<U12'
PADDING_TOKEN = 'PAD'
LIMIT_OF_TOKENS = 100000
```

In [0]:

```
from nltk import word_tokenize
import numpy as np
import string
import re

def tokenize_string(_string):
    return [tok_.lower() for tok_ in word_tokenize(_string)
            if not re.fullmatch('[' + string.punctuation + ']+', tok_)]

def pad(A, length):
    arr = np.empty(length, dtype = STRING_DTYPE)
    arr.fill(PADDING_TOKEN)
    arr[:len(A)] = A
    return arr

def tokenize_row(_sentence):
    return pad(tokenize_string(_sentence)[:MAX_LENGTH], MAX_LENGTH)

def encode_row(_label):
    return 1 if _label == 'positive' else 0

def encode_and_tokenize(_dataframe):

    tttt = _dataframe.apply(lambda row: tokenize_row(row['review']), axis = 1)
    llll = _dataframe.apply(lambda row: encode_row(row['sentiment']), axis = 1)

    data_dict_ = { 'label': llll, 'tokens': tttt }

    encoded_and_tokenized_ = pd.DataFrame(data_dict_,
                                           columns = [ 'label', 'tokens' ])

    return encoded_and_tokenized_
```

In [0]:

```
df_train_tokenized = encode_and_tokenize(df_train)
df_test_tokenized = encode_and_tokenize(df_test)
```

In [0]:

```
from collections import Counter

def get_tokens_list(_dataframe):

    all_tokens_ = []

    for sent_ in _dataframe['tokens'].values:
        all_tokens_.extend(sent_)

    tokens_counter_ = Counter(all_tokens_)

    return [t for t, _ in tokens_counter_.most_common(LIMIT_OF_TOKENS)]
```

In [0]:

```
tokens_list = get_tokens_list(pd.concat([df_train_tokenized,
                                         df_test_tokenized]))
```

In [0]:

```
word_to_int_dict = {}

word_to_int_dict.update(
    {t : i for i, t in enumerate(tokens_list)})
```

In [0]:

```
def intize_row(_tokens):
    return np.array([word_to_int_dict[t]
                    if t in word_to_int_dict
                    else 0
                    for t in _tokens])

def encode_and_tokenize(_dataframe):

    iii = _dataframe.apply(lambda row: intize_row(row['tokens']), axis = 1)

    data_dict_ = { 'label': _dataframe['label'], 'ints': iii }

    intized_ = pd.DataFrame(data_dict_, columns = ['label', 'ints'])

    return intized_
```

In [0]:

```
df_train_intized = encode_and_tokenize(df_train_tokenized)
df_test_intized = encode_and_tokenize(df_test_tokenized)
```

## Задание 2

Реализуйте и обучите двунаправленную рекуррентную сеть (*LSTM* или *GRU*).

Какого качества классификации удалось достичь?

In [0]:

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

In [0]:

```
import tensorflow as tf

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

from tensorflow import keras
```

In [0]:

```
# To fix memory Leak: https://github.com/tensorflow/tensorflow/issues/33009

tf.compat.v1.disable_eager_execution()
```

Здесь будем использовать такую конфигурацию рекуррентного *LSTM*-слоя, которая позволит использовать очень быструю *cuDNN* имплементацию.

In [0]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import Bidirectional, LSTM, Dense

# The requirements to use the cuDNN implementation are:
# 1. `activation` == `tanh`
# 2. `recurrent_activation` == `sigmoid`
# 3. `recurrent_dropout` == 0
# 4. `unroll` is `False`
# 5. `use_bias` is `True`
# 6. `reset_after` is `True`
# 7. Inputs, if use masking, are strictly right-padded.

model = tf.keras.Sequential()

model.add(Bidirectional(LSTM(100, return_sequences = False),
                       merge_mode = 'concat', input_shape = (MAX_LENGTH, 1)))
model.add(Dense(1, activation = 'sigmoid'))
```

In [0]:

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

```
model.summary()
```

Model: "sequential"

Layer (type)	Output Shape	Param #
bidirectional (Bidirectional (None, 200))		81600
dense (Dense)	(None, 1)	201
<hr/>		
Total params: 81,801		
Trainable params: 81,801		
Non-trainable params: 0		

In [0]:

```
X_train_intized = np.asarray(list(df_train_intized['ints'].values),
                           dtype = float)[..., np.newaxis]
X_test_intized = np.asarray(list(df_test_intized['ints'].values),
                           dtype = float)[..., np.newaxis]

y_train_intized = np.asarray(list(df_train_intized['label'].values))
y_test_intized = np.asarray(list(df_test_intized['label'].values))
```

In [0]:

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 8, 6

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_accuracy(_history,
                  _train_acc_name = 'accuracy',
                  _val_acc_name = 'val_accuracy'):

    plt.plot(_history.history[_train_acc_name])
    plt.plot(_history.history[_val_acc_name])

    plt.title('Model accuracy')

    plt.ylabel('Accuracy')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

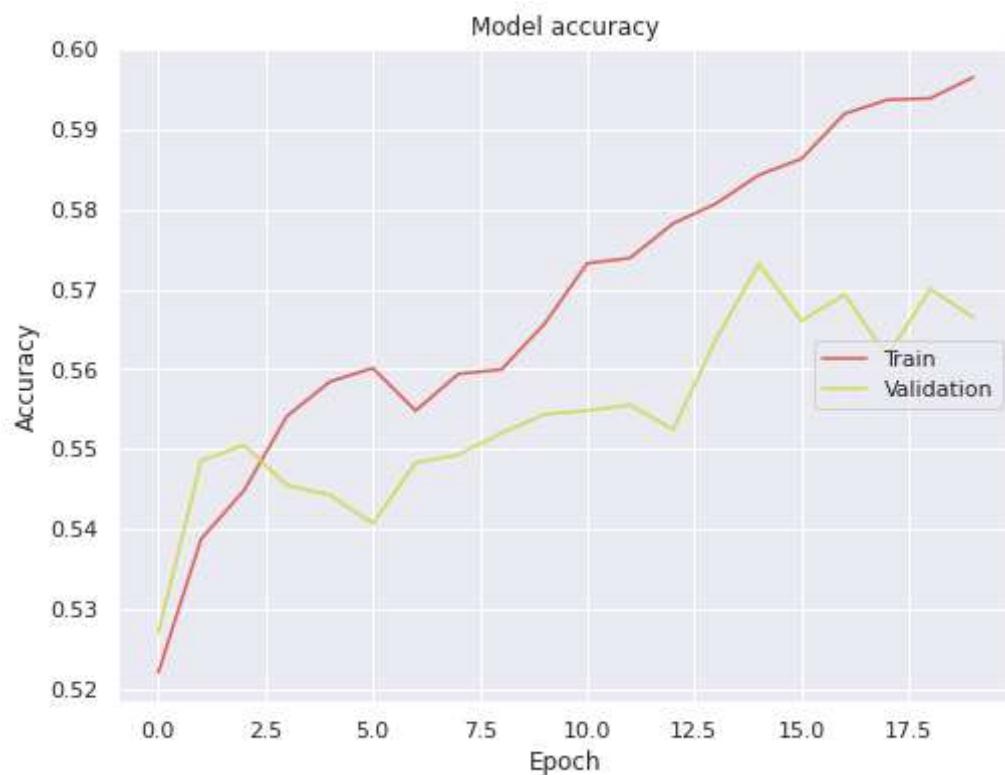
    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

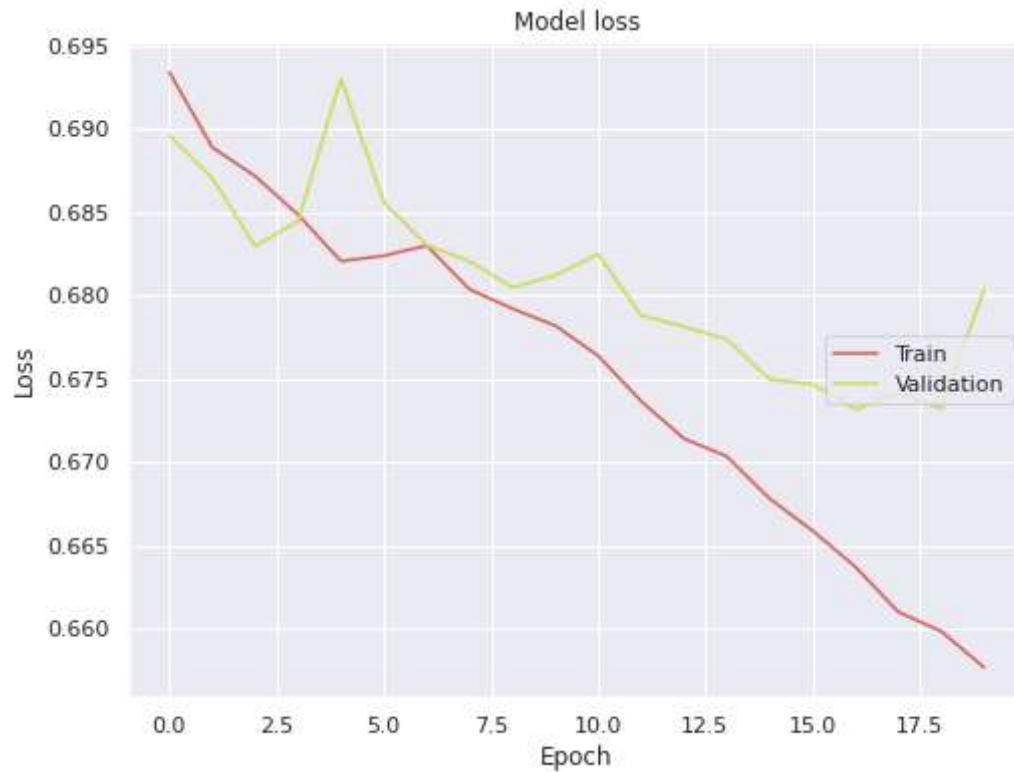
In [0]:

```
plot_accuracy(history)
```



In [0]:

```
plot_loss(history)
```



In [0]:

```
results = model.evaluate(X_test_intized, y_test_intized)
print('Test loss, test accuracy:', results)
```

Test loss, test accuracy: [0.6782044233322143, 0.571]

На валидационной выборке удалось достичь точности 57%.

### Задание 3

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

Используем 300-мерные вектора *FastText* — лучшую на сегодняшний день имплементацию word2vec:  
<https://fasttext.cc/docs/en/english-vectors.html> (<https://fasttext.cc/docs/en/english-vectors.html>). Файл пришлось доработать — 9-я строка не читалась.

In [0]:

```
# VECTORS_ARCHIVE_NAME = 'wiki-news-300d-1M-fixed.zip'  
  
# VECTORS_FILE_NAME = 'wiki-news-300d-1M-fixed.vec'  
  
# VECTORS_LOCAL_DIR_NAME = 'vectors'
```

In [0]:

```
# with ZipFile(os.path.join(BASE_DIR, VECTORS_ARCHIVE_NAME), 'r') as zip_:  
#     zip_.extractall(VECTORS_LOCAL_DIR_NAME)
```

Создадим уменьшенный словарь, содержащий только встреченные токены, чтобы уменьшить нагрузку на Google Drive:

In [0]:

```
# def build_vectors_dict(_actual_tokens, _vectors_file_path,  
#                         _unknown_token = 'unknown'):  
  
#     vec_data_ = pd.read_csv(_vectors_file_path, sep = ' ',  
#                            header = None, skiprows = [9])  
  
#     actual_vectors_ = [x for x in vec_data_.values  
#                        if x[0] in _actual_tokens or x[0] == _unknown_token]  
  
#     return actual_vectors_
```

In [0]:

```
# actual_vectors = build_vectors_dict(tokens_list,  
#                                       os.path.join(VECTORS_LOCAL_DIR_NAME,  
#                                                   VECTORS_FILE_NAME))
```

In [0]:

```
# vectors_np = np.array(actual_vectors)  
  
# vectors_dict = dict(zip(vectors_np[:, 0], vectors_np[:, 1:]))  
  
# vectors_dict_file_name = 'word-vec-dict-{}-items'.format(len(vectors_dict))  
  
# vectors_dict_file_path = os.path.join(BASE_DIR, vectors_dict_file_name)  
  
# np.savez_compressed(vectors_dict_file_path, vectors_dict, allow_pickle = True)
```

In [0]:

```
vectors_dict_file_path = (  
    './drive/My Drive/Colab Files/mo-2/word-vec-dict-56485-items.npz')
```

In [0]:

```
vectors_dict_data = np.load(vectors_dict_file_path, allow_pickle = True)
vectors_dict = vectors_dict_data['arr_0'][()]
```

In [0]:

```
VECTORS_LENGTH = 300
```

In [0]:

```
def tokens_to_vectors(_word_to_vec_dict, _tokens, _unknown_token):
    return [_word_to_vec_dict[t]
            if t in _word_to_vec_dict
            else _word_to_vec_dict[_unknown_token]
            for t in _tokens]

def row_to_vectors(_tokens):
    return np.array(tokens_to_vectors(vectors_dict, _tokens, 'unknown'))

def vectorize(_dataframe):
    vvvv = _dataframe.apply(lambda row: row_to_vectors(row['tokens']), axis = 1)
    data_dict_ = { 'label': _dataframe['label'], 'vectors': vvvv }
    vectorized_ = pd.DataFrame(data_dict_, columns = ['label', 'vectors'])

    return vectorized_
```

In [0]:

```
df_train_vectorized = vectorize(df_train_tokenized)
df_test_vectorized = vectorize(df_test_tokenized)
```

In [0]:

```
X_train_vectorized = np.asarray(list(df_train_vectorized['vectors'].values),
                                 dtype = float)
X_test_vectorized = np.asarray(list(df_test_vectorized['vectors'].values),
                               dtype = float)

y_train_vectorized = np.asarray(list(df_train_vectorized['label'].values))
y_test_vectorized = np.asarray(list(df_test_vectorized['label'].values))
```

In [0]:

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

model_2.add(Bidirectional(LSTM(100, return_sequences = False),
                         merge_mode = 'concat',
                         input_shape = (MAX_LENGTH, VECTORS_LENGTH)))
model_2.add(Dense(1, activation = 'sigmoid'))
```

In [0]:

```
model_2.compile(optimizer = 'adam',
                 loss = 'binary_crossentropy',
                 metrics = ['accuracy'])

model_2.summary()
```

Model: "sequential\_1"

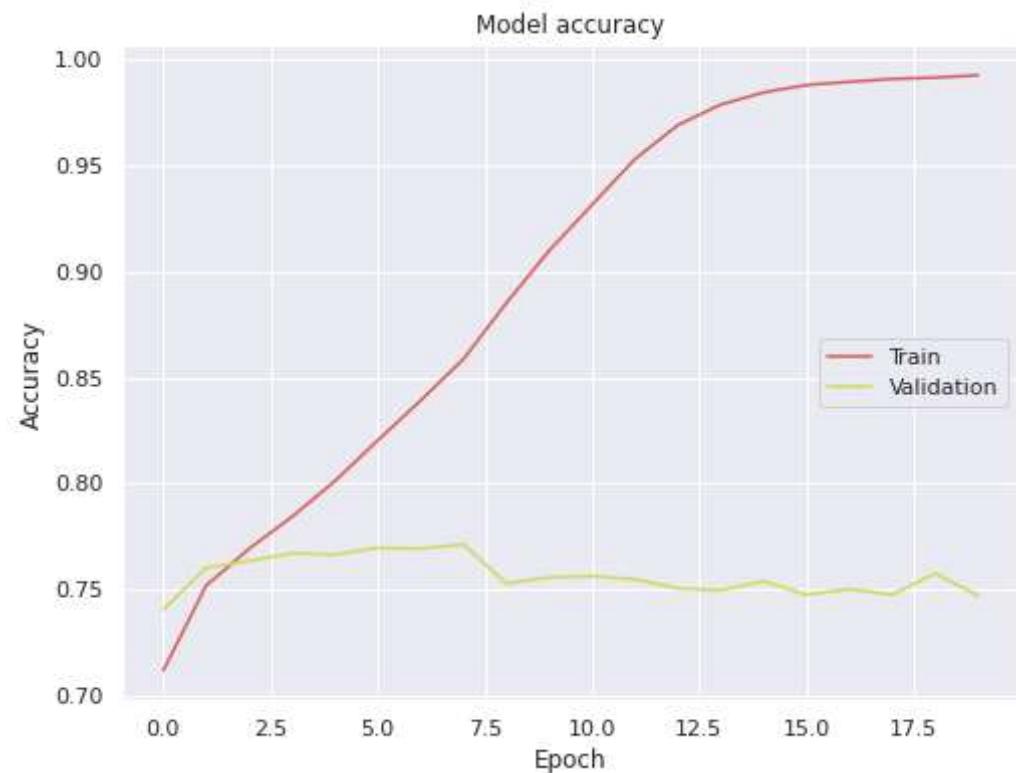
Layer (type)	Output Shape	Param #
bidirectional_1 (Bidirection (None, 200)		320800
dense_1 (Dense)	(None, 1)	201
Total params:	321,001	
Trainable params:	321,001	
Non-trainable params:	0	

In [0]:

```
history_2 = model_2.fit(x = X_train_vectorized, y = y_train_vectorized,
                         epochs = 20, validation_split = 0.15, verbose = 0)
```

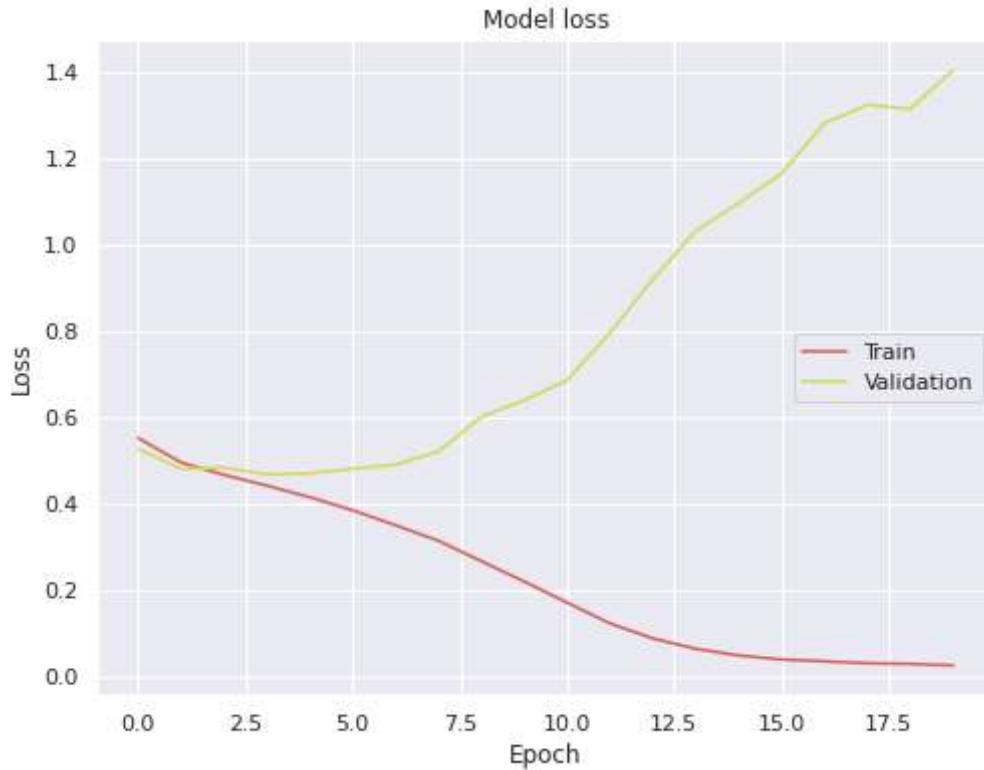
In [44]:

```
plot_accuracy(history_2)
```



In [45]:

```
plot_loss(history_2)
```



In [46]:

```
results_2 = model_2.evaluate(X_test_vectorized, y_test_vectorized)  
print('Test loss, test accuracy:', results_2)
```

Test loss, test accuracy: [1.384592835521698, 0.7472]

Как и ожидалось, использование эмбеддингов показало лучший результат, чем кодирование слов просто целыми числами — 74%.

## Задание 4

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

In [0]:

```
model_3 = tf.keras.Sequential()

model_3.add(Bidirectional(LSTM(5, return_sequences = True),
                         merge_mode = 'concat',
                         input_shape = (MAX_LENGTH, VECTORS_LENGTH)))
model_3.add(LSTM(1, return_sequences = False))
model_3.add(Dense(10, activation = 'linear'))
model_3.add(Dense(1, activation = 'sigmoid'))
```

In [48]:

```
model_3.compile(optimizer = 'adam',
                 loss = 'binary_crossentropy',
                 metrics = ['accuracy'])

model_3.summary()
```

Model: "sequential\_2"

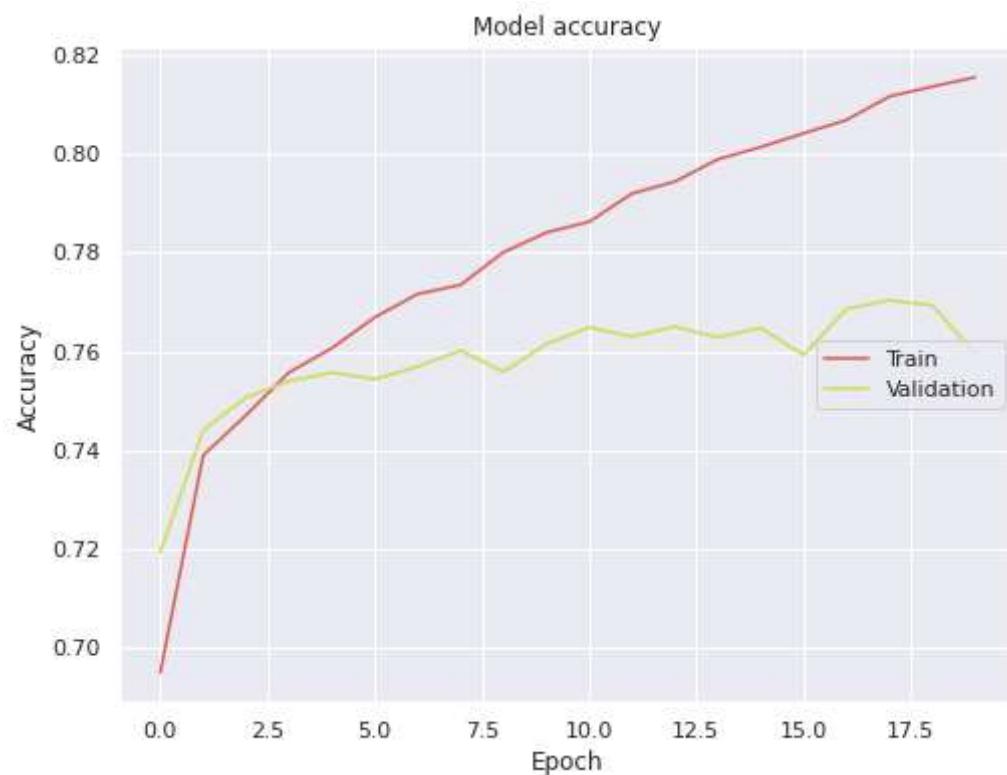
Layer (type)	Output Shape	Param #
=====		
bidirectional_2 (Bidirection (None, 40, 10)		12240
lstm_3 (LSTM)	(None, 1)	48
dense_2 (Dense)	(None, 10)	20
dense_3 (Dense)	(None, 1)	11
=====		
Total params:	12,319	
Trainable params:	12,319	
Non-trainable params:	0	

In [0]:

```
history_3 = model_3.fit(x = X_train_vectorized, y = y_train_vectorized,
                         validation_split = 0.15, epochs = 20, verbose = 0)
```

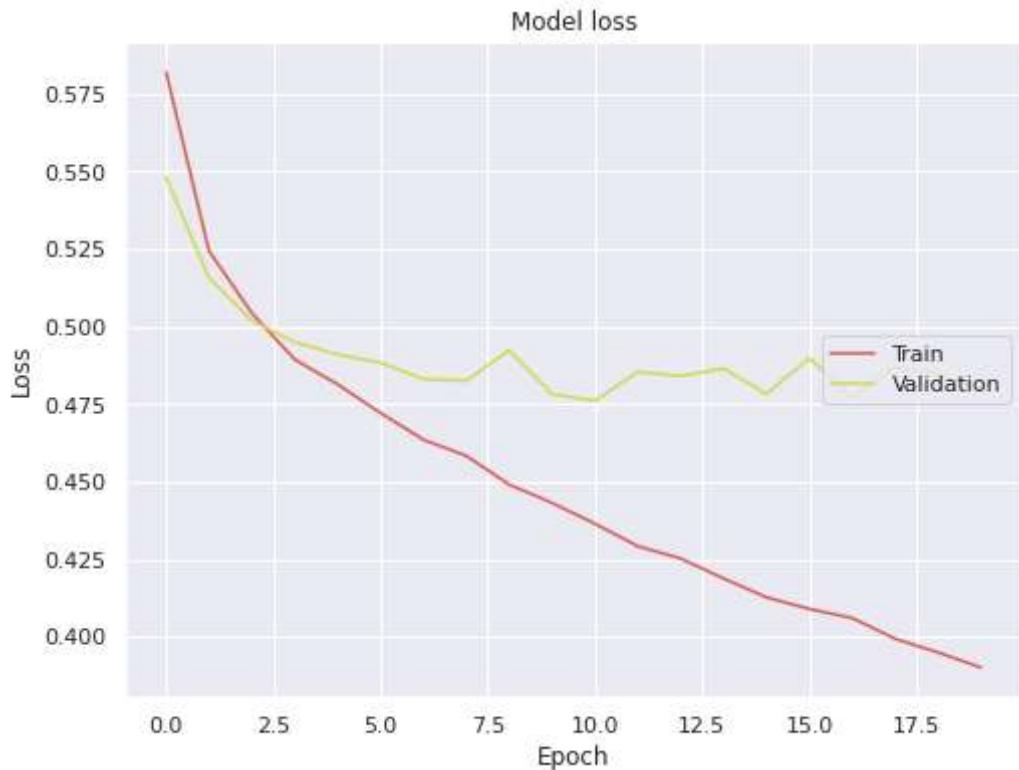
In [50]:

```
plot_accuracy(history_3)
```



In [51]:

```
plot_loss(history_3)
```



In [52]:

```
results_3 = model_3.evaluate(X_test_vectorized, y_test_vectorized)  
print('Test loss, test accuracy:', results_3)
```

Test loss, test accuracy: [0.4730978265762329, 0.7682]

Добавление ещё одного рекуррентного слоя ненамного улучшило результат — точность 76% на тестовой выборке.

## Задание 5

Используйте предобученную рекуррентную нейронную сеть (например, *DeepMoji* или что-то подобное).

Какой максимальный результат удалось получить на контрольной выборке?

На своих моделях удалось достичнуть максимальной точности 76% на тестовой выборке.

# Лабораторная работа №8

## Рекуррентные нейронные сети для анализа временных рядов

Набор данных для прогнозирования временных рядов, который состоит из среднемесячного числа пятен на солнце, наблюдавшихся 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/>)

### Задание 1

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

In [0]:

```
import warnings  
warnings.filterwarnings('ignore')
```

In [2]:

```
from google.colab import drive  
drive.mount('/content/drive', force_remount = True)
```

Mounted at /content/drive

In [0]:

```
BASE_DIR = '/content/drive/My Drive/Colab Files/mo-2'  
  
import sys  
  
sys.path.append(BASE_DIR)  
  
import os
```

In [0]:

```
DATA_ARCHIVE_NAME = 'sunspots.zip'  
  
LOCAL_DIR_NAME = 'sunspots'
```

In [0]:

```
from zipfile import ZipFile

with ZipFile(os.path.join(BASE_DIR, DATA_ARCHIVE_NAME), 'r') as zip_:
    zip_.extractall(LOCAL_DIR_NAME)
```

In [0]:

```
DATA_FILE_PATH = 'sunspots/Sunspots.csv'
```

In [0]:

```
import pandas as pd

all_df = pd.read_csv(DATA_FILE_PATH, parse_dates = ['Date'], index_col = 'Date')
```

In [8]:

```
print(all_df.shape)
```

(3252, 2)

In [9]:

```
all_df.keys()
```

Out[9]:

Index(['Unnamed: 0', 'Monthly Mean Total Sunspot Number'], dtype='object')

In [0]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

additive = seasonal_decompose(all_df['Monthly Mean Total Sunspot Number'],
                               model = 'additive', extrapolate_trend = 'freq')
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
import seaborn as sns
from matplotlib import rcParams

rcParams['figure.figsize'] = 12, 8

sns.set()
sns.set_palette(sns.color_palette('hls'))

def plot_loss(_history):

    plt.plot(_history.history['loss'])
    plt.plot(_history.history['val_loss'])

    plt.title('Model loss')

    plt.ylabel('Loss')
    plt.xlabel('Epoch')

    plt.legend(['Train', 'Validation'], loc = 'right')

    plt.show()
```

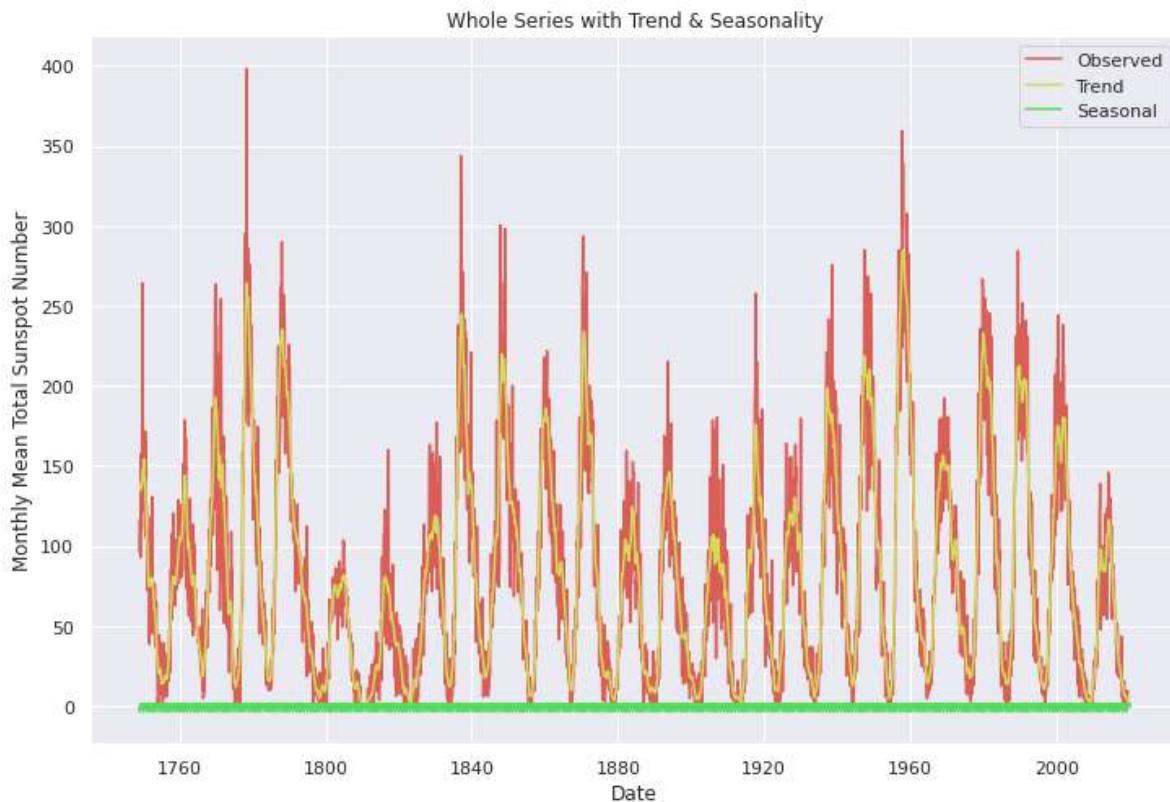
In [12]:

```
sns.lineplot(data = additive.observed, label = 'Observed')
sns.lineplot(data = additive.trend, label = 'Trend')
sns.lineplot(data = additive.seasonal, label = 'Seasonal')

plt.xlabel('Date')
plt.ylabel('Monthly Mean Total Sunspot Number')

plt.title('Whole Series with Trend & Seasonality')

plt.show()
```



Рассмотрим подробнее на небольшом промежутке:

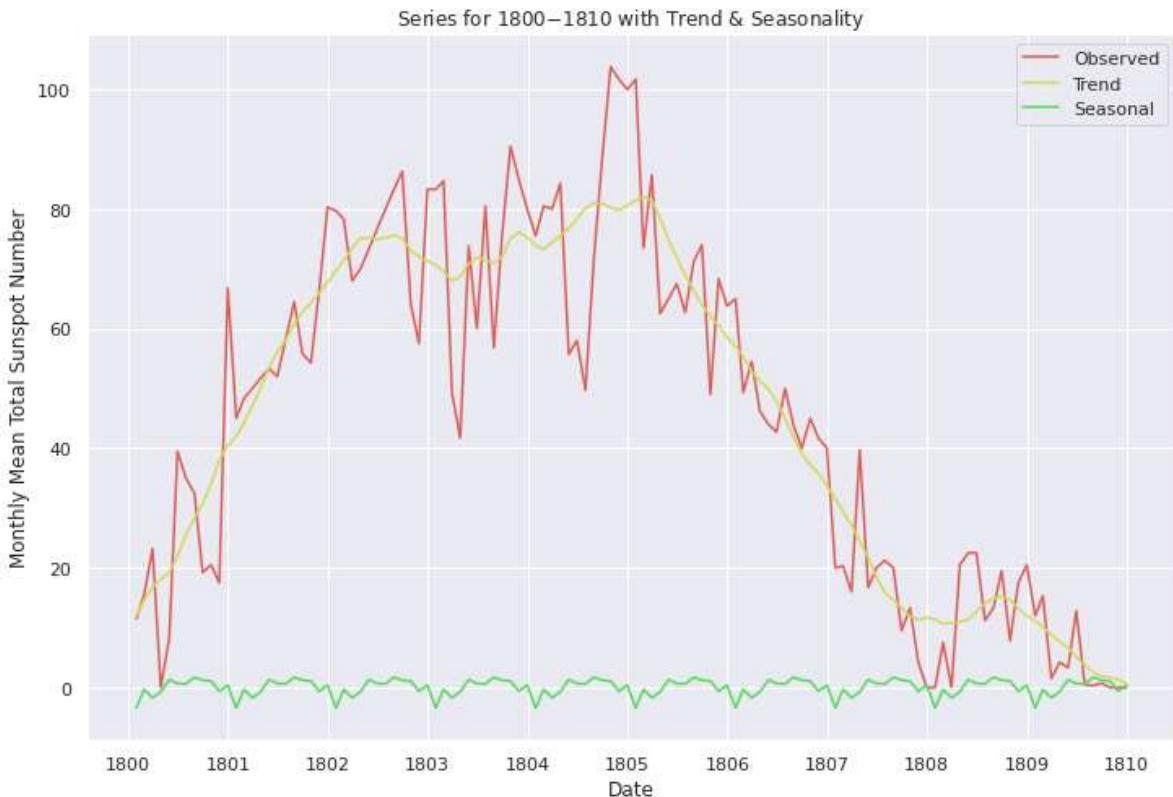
In [13]:

```
sns.lineplot(data = additive.observed['1800-01-01':'1810-01-01'], label = 'Observed')
sns.lineplot(data = additive.trend['1800-01-01':'1810-01-01'], label = 'Trend')
sns.lineplot(data = additive.seasonal['1800-01-01':'1810-01-01'], label = 'Seasonal')

plt.xlabel('Date')
plt.ylabel('Monthly Mean Total Sunspot Number')

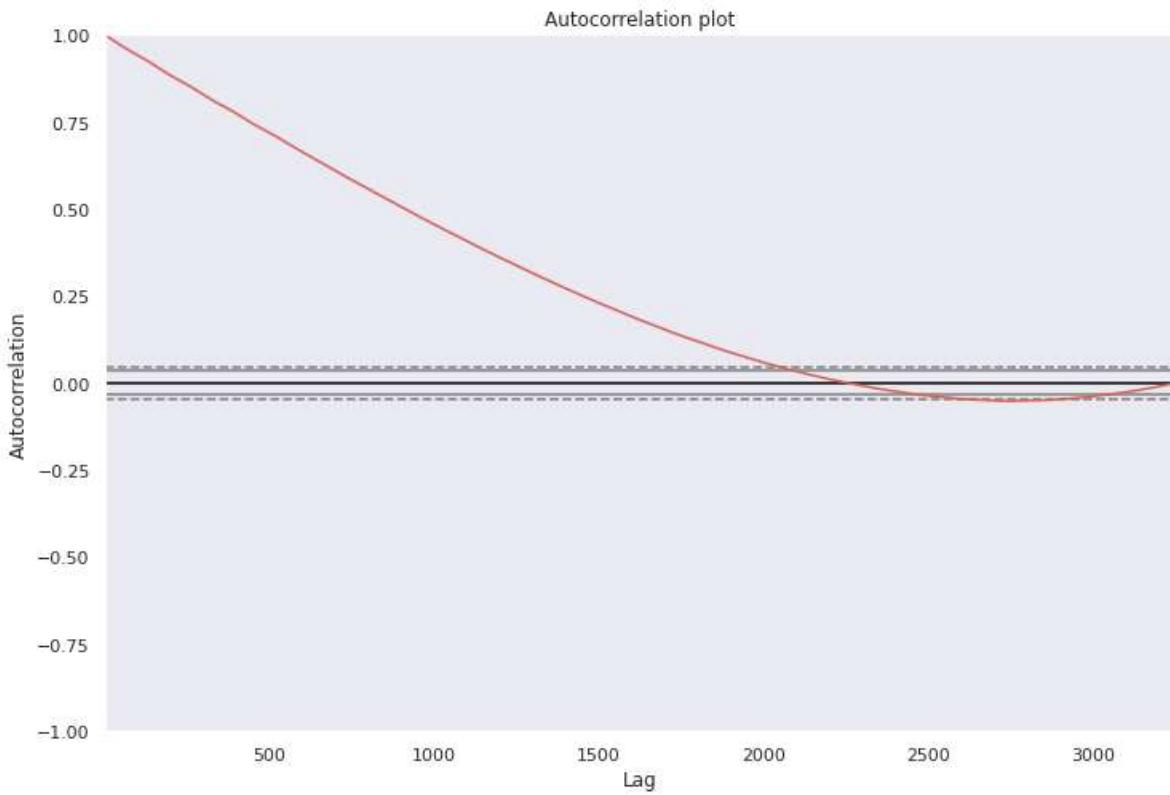
plt.title('Series for 1800$-$1810 with Trend & Seasonality')

plt.show()
```



In [14]:

```
from pandas.plotting import autocorrelation_plot  
  
autocorrelation_plot(all_df.values.tolist())  
  
plt.title('Autocorrelation plot')  
  
plt.show()
```



## Задание 2

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

Этот шаг будет применён автоматически с помощью индексации массива данных и как параметр `validation_split` метода `model.fit()`.

## Задание 3

Примените модель ARIMA для прогнозирования значений данного временного ряда.

In [0]:

```
! pip install pmdarima --quiet
```

In [16]:

```
from statsmodels.tsa.arima_model import ARIMA

model = ARIMA(all_df['Monthly Mean Total Sunspot Number'].values,
              order = (1, 0, 2))

model_fit = model.fit(disp = 0)

print(model_fit.summary())
```

### ARMA Model Results

```
=====
==                                         Dep. Variable:                  y
No. Observations:                      32
52
Model:                             ARMA(1, 2)   Log Likelihood      -15098.8
79
Method:                            css-mle    S.D. of innovations     25.1
20
Date:        Mon, 20 Apr 2020    AIC            30207.7
57
Time:          04:10:50      BIC            30238.1
92
Sample:             0      HQIC           30218.6
60

=====
```

```
==                                         coef      std err       z      P>|z|      [0.025      0.97
5]
-----
-- const      81.1334     11.977     6.774      0.000     57.660     104.6
07
ar.L1.y      0.9826     0.003     284.234      0.000      0.976      0.9
89
ma.L1.y     -0.4063     0.018    -22.947      0.000     -0.441     -0.3
72
ma.L2.y     -0.1140     0.017     -6.894      0.000     -0.146     -0.0
82
```

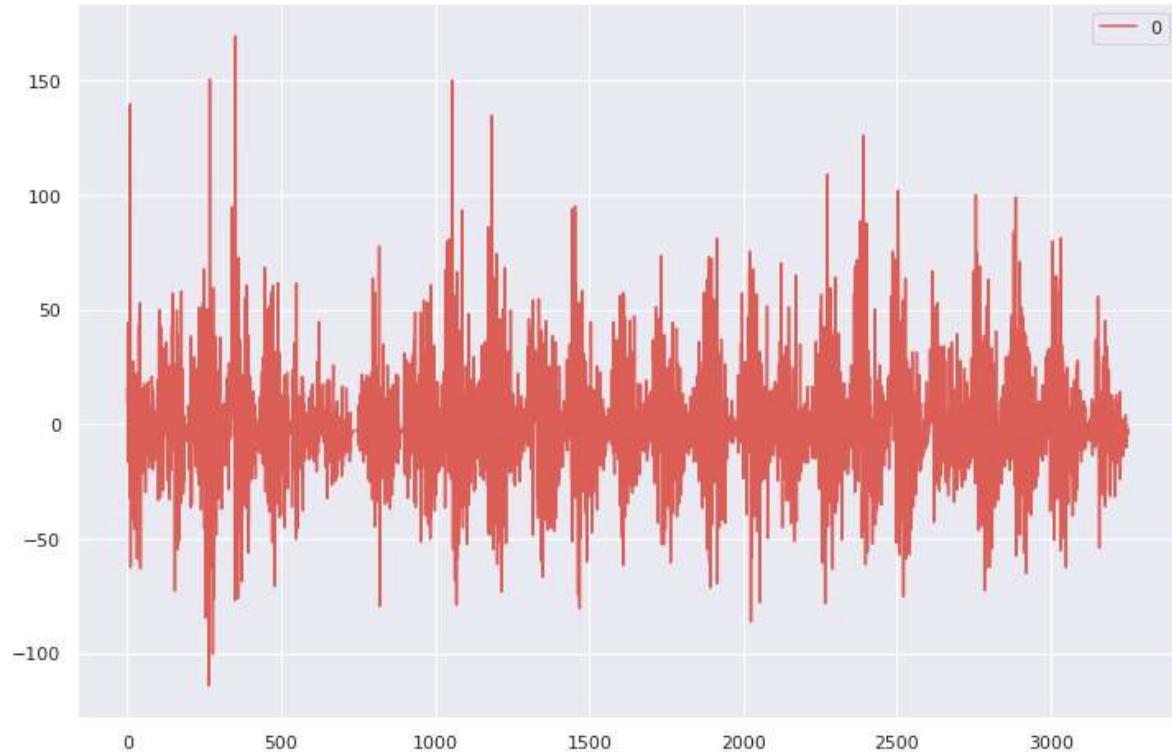
### Roots

```
=====
=                                         Real      Imaginary      Modulus      Frequenc
y
-----
- AR.1      1.0177      +0.0000j      1.0177      0.000
0
MA.1       1.6743      +0.0000j      1.6743      0.000
0
MA.2      -5.2373      +0.0000j      5.2373      0.500
0
```



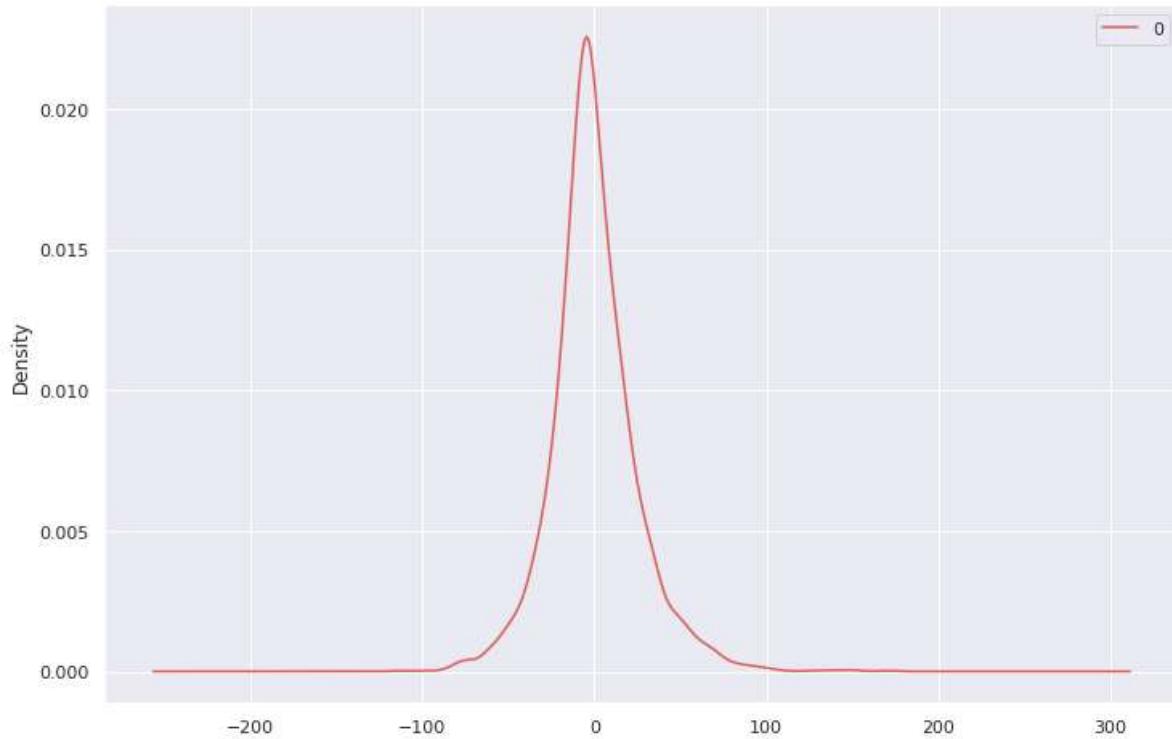
In [17]:

```
residuals = pd.DataFrame(model_fit.resid)  
residuals.plot()  
plt.show()
```



In [18]:

```
residuals.plot(kind = 'kde')  
plt.show()
```



## Задание 4

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

Сначала нужно создать датасет из данных.

In [0]:

```
TEST_PERIOD = 600
```

In [0]:

```
OBSERVATIONS_PER_CYCLE = 11 * 12
```

In [0]:

```
TIME_STEPS = OBSERVATIONS_PER_CYCLE
```

In [0]:

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

In [0]:

```
import tensorflow as tf

tf.compat.v1.logging.set_verbosity(tf.compat.v1.logging.ERROR)

from tensorflow import keras
```

In [0]:

```
import numpy as np
from datetime import timezone

def timeseries_to_dataset(_X_ts, _time_steps):

    samples_n_ = len(_X_ts) - _time_steps

    _X_norm = tf.keras.utils.normalize(_X_ts).squeeze()

    print(_X_ts.shape, _X_norm.shape)

    X_ = np.zeros((samples_n_, _time_steps))
    y_ = np.zeros((samples_n_,))

    for i in range(samples_n_):

        X_[i] = _X_norm[i:(i + _time_steps)]
        y_[i] = _X_norm[(i + _time_steps)]

    return X_[..., np.newaxis], y_
```

In [25]:

```
X_as_ds, y_as_ds = timeseries_to_dataset(
    all_df['Monthly Mean Total Sunspot Number'].values,
    TIME_STEPS)

X, y = X_as_ds[:TEST_PERIOD], y_as_ds[:TEST_PERIOD]

X_test, y_test = X_as_ds[-TEST_PERIOD:], y_as_ds[-TEST_PERIOD:]

(3252,), (3252,)
```

In [26]:

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

```
(2520, 132, 1) (600, 132, 1) (2520,) (600,)
```

In [0]:

```
from tensorflow.keras.models import Sequential
from tensorflow.keras.layers import LSTM, Dense

model = tf.keras.Sequential()

model.add(LSTM(2, activation = 'relu', return_sequences = True,
              input_shape = X.shape[-2:]))
model.add(LSTM(12, activation = 'relu'))
model.add(Dense(1))
```

In [28]:

```
model.compile(optimizer = 'adam',
              loss = 'mse')

model.summary()
```

Model: "sequential"

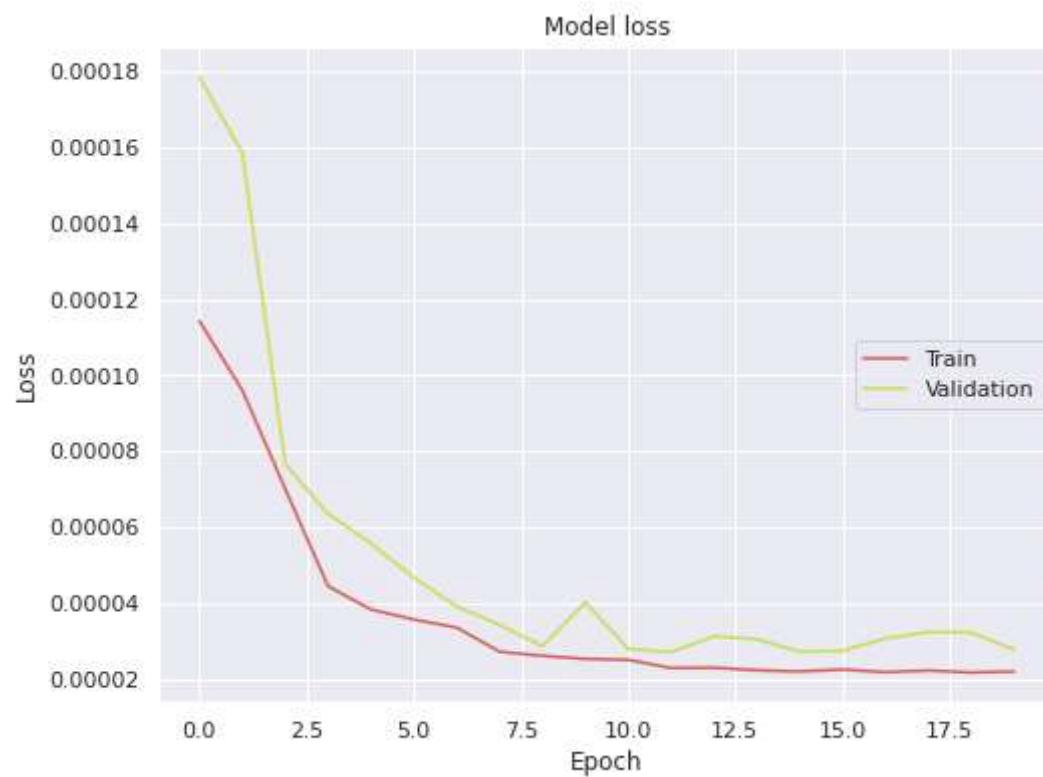
Layer (type)	Output Shape	Param #
lstm (LSTM)	(None, 132, 2)	32
lstm_1 (LSTM)	(None, 12)	720
dense (Dense)	(None, 1)	13
Total params:	765	
Trainable params:	765	
Non-trainable params:	0	

In [0]:

```
history = model.fit(x = X, y = y, epochs = 20, validation_split = 0.15,
                     verbose = 0)
```

In [30]:

```
rcParams['figure.figsize'] = 8, 6  
plot_loss(history)
```



In [31]:

```
results = model.evaluate(X_test, y_test)  
print('Test mse:', results)
```

```
19/19 [=====] - 0s 26ms/step - loss: 1.9438e-05  
Test mse: 1.9438315575825982e-05
```

In [32]:

```
y_pred = model.predict(X_test[20][np.newaxis, ...])
```

```
print(y_pred, y_test[20])
```

```
[[0.01421428]] 0.011741172416758197
```

## Задание 5

Сравните качество прогноза моделей.

Какой максимальный результат удалось получить на контрольной выборке?