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

Минск 2020

Лабораторная работа №1

Логистическая регрессия в качестве нейронной сети

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

Данные можно скачать по ссылке:

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

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

Задание 1

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

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
```

In [3]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

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 = (16, 6))

    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]:

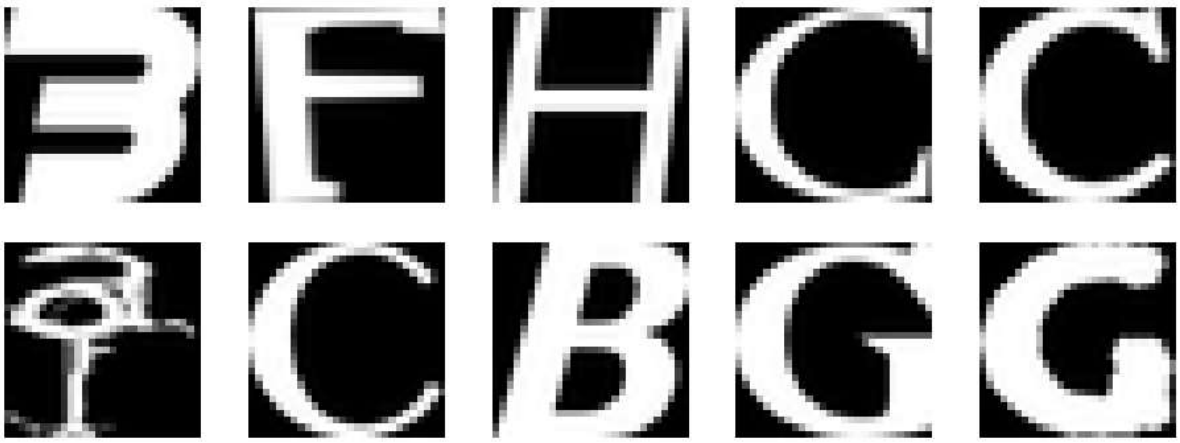
```
def tar_to_dataframe(_tar_url, _key):

    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 [24]:

```
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_
```

```
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:94
0: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:94
0: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
/usr/local/lib/python3.6/dist-packages/sklearn/linear_model/_logistic.py:94
0: ConvergenceWarning: lbfgs failed to converge (status=1):
STOP: TOTAL NO. of ITERATIONS REACHED LIMIT.

Increase the number of iterations (max_iter) or scale the data as shown in:
    https://scikit-learn.org/stable/modules/preprocessing.html (https://scik
it-learn.org/stable/modules/preprocessing.html)
Please also refer to the documentation for alternative solver options:
    https://scikit-learn.org/stable/modules/linear_model.html#logistic-regre
ssion (https://scikit-learn.org/stable/modules/linear_model.html#logistic-re
gression)
  extra_warning_msg=_LOGISTIC_SOLVER_CONVERGENCE_MSG)
```

In [25]:

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

```
C	C	B	H	C	G	B	E	G	H
```

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)
```

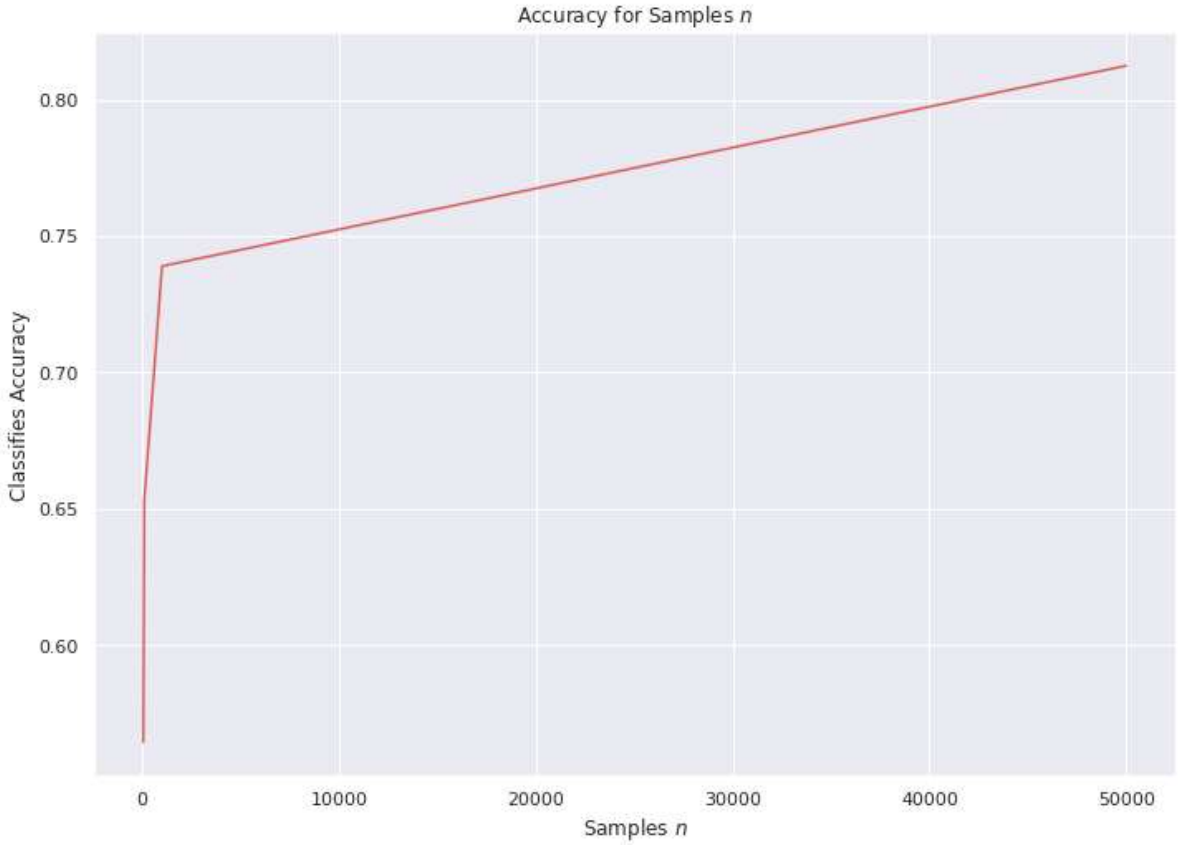
In [28]:

```
print(scores)
```

```
{50: 0.5643, 100: 0.6529, 1000: 0.7389, 50000: 0.8124}
```

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 28×28 первых 10 букв латинского алфавита (*A_ ... _J*, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке:

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

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

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.).

In [1]:

```
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 [4]:

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

Out[4]:

```
(461946,)
```

In [5]:

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

! pip show tensorflow-gpu
```

```
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: absl-py, keras-preprocessing, termcolor, opt-einsum, grpcio, proto
buf, google-pasta, tensorflow-estimator, wrapt, six, wheel, scipy, tensorboa
rd, numpy, h5py, gast, astunparse
Required-by:
```

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 [9]:

```
x = np.asarray(list(dataframe['data']))[..., np.newaxis]

x = tf.keras.utils.normalize(x, axis = 1)

x.shape
```

Out[9]:

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

In [10]:

```
x_test = np.asarray(list(dataframe_test['data']))[..., np.newaxis]

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

x_test.shape
```

Out[10]:

```
(46195, 28, 28, 1)
```

In [0]:

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

In [12]:

```
from tensorflow.keras.utils import to_categorical

y = to_categorical(dataframe['label'].astype('category').cat.codes.astype('int32'))

y.shape
```

Out[12]:

```
(415751, 10)
```

In [13]:

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

y_test.shape
```

Out[13]:

```
(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 [18]:

```
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 [21]:

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

```
Epoch 1/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2191
- categorical_accuracy: 0.1130 - val_loss: 4.8881 - val_categorical_accurac
y: 0.0000e+00
Epoch 2/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2036
- categorical_accuracy: 0.1121 - val_loss: 5.5146 - val_categorical_accurac
y: 0.0000e+00
Epoch 3/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2018
- categorical_accuracy: 0.1131 - val_loss: 5.8773 - val_categorical_accurac
y: 0.0000e+00
Epoch 4/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2010
- categorical_accuracy: 0.1135 - val_loss: 6.1279 - val_categorical_accurac
y: 0.0000e+00
Epoch 5/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2007
- categorical_accuracy: 0.1128 - val_loss: 6.3155 - val_categorical_accurac
y: 0.0000e+00
Epoch 6/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2005
- categorical_accuracy: 0.1125 - val_loss: 6.4637 - val_categorical_accurac
y: 0.0000e+00
Epoch 7/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2003
- categorical_accuracy: 0.1131 - val_loss: 6.5846 - val_categorical_accurac
y: 0.0000e+00
Epoch 8/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2002
- categorical_accuracy: 0.1126 - val_loss: 6.6856 - val_categorical_accurac
y: 0.0000e+00
Epoch 9/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2002
- categorical_accuracy: 0.1130 - val_loss: 6.7715 - val_categorical_accurac
y: 0.0000e+00
Epoch 10/10
11693/11693 [==============================] - 102s 9ms/step - loss: 2.2001
- categorical_accuracy: 0.1130 - val_loss: 6.8451 - val_categorical_accurac
y: 0.0000e+00
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

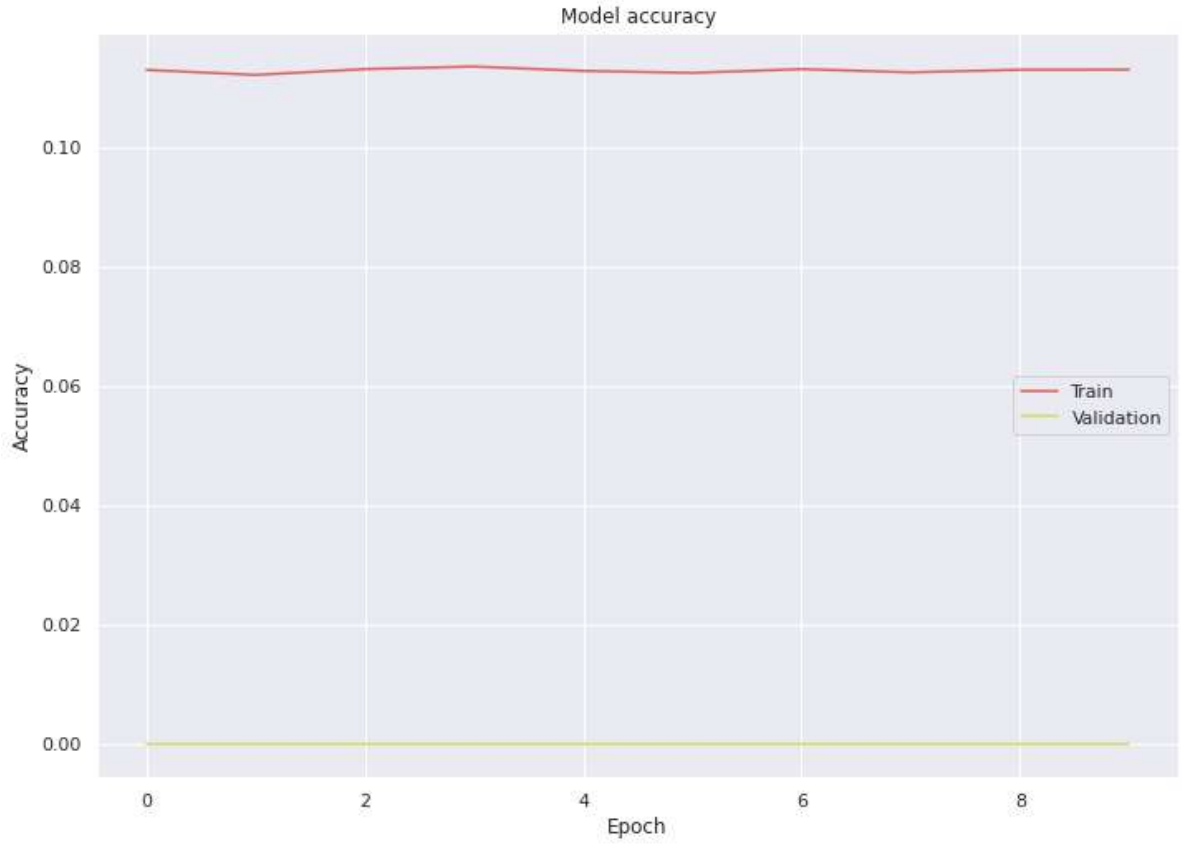
In [23]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

In [24]:

```
import matplotlib.pyplot as plt

plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

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 [25]:

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

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

```
1444/1444 [==============================] - 5s 3ms/step - loss: 2.6747 - ca
tegorical_accuracy: 0.1050
Test loss, test accuracy: [2.6747310161590576, 0.10496807098388672]
```

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,
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_cross_from_logits,
                metrics = ['categorical_accuracy'])
```

In [32]:

```
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
_________________________________________________________________
```

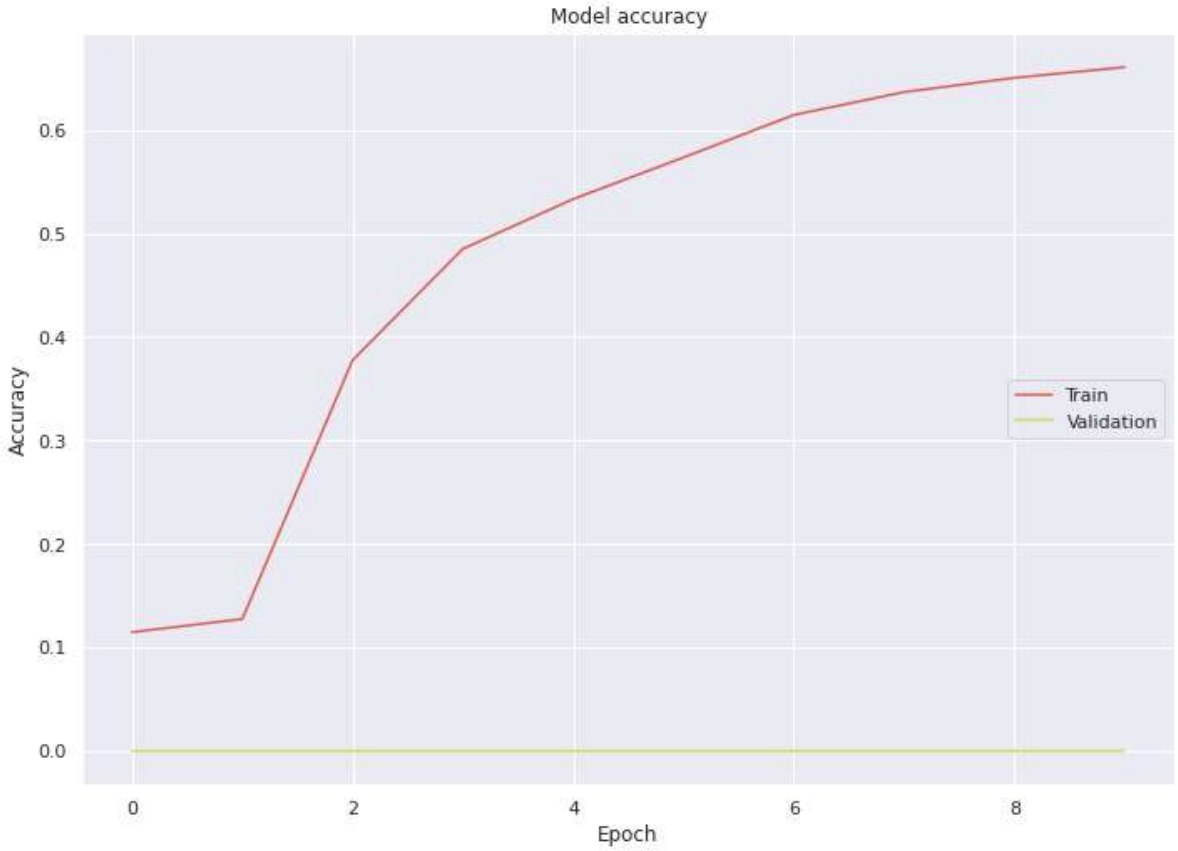
In [33]:

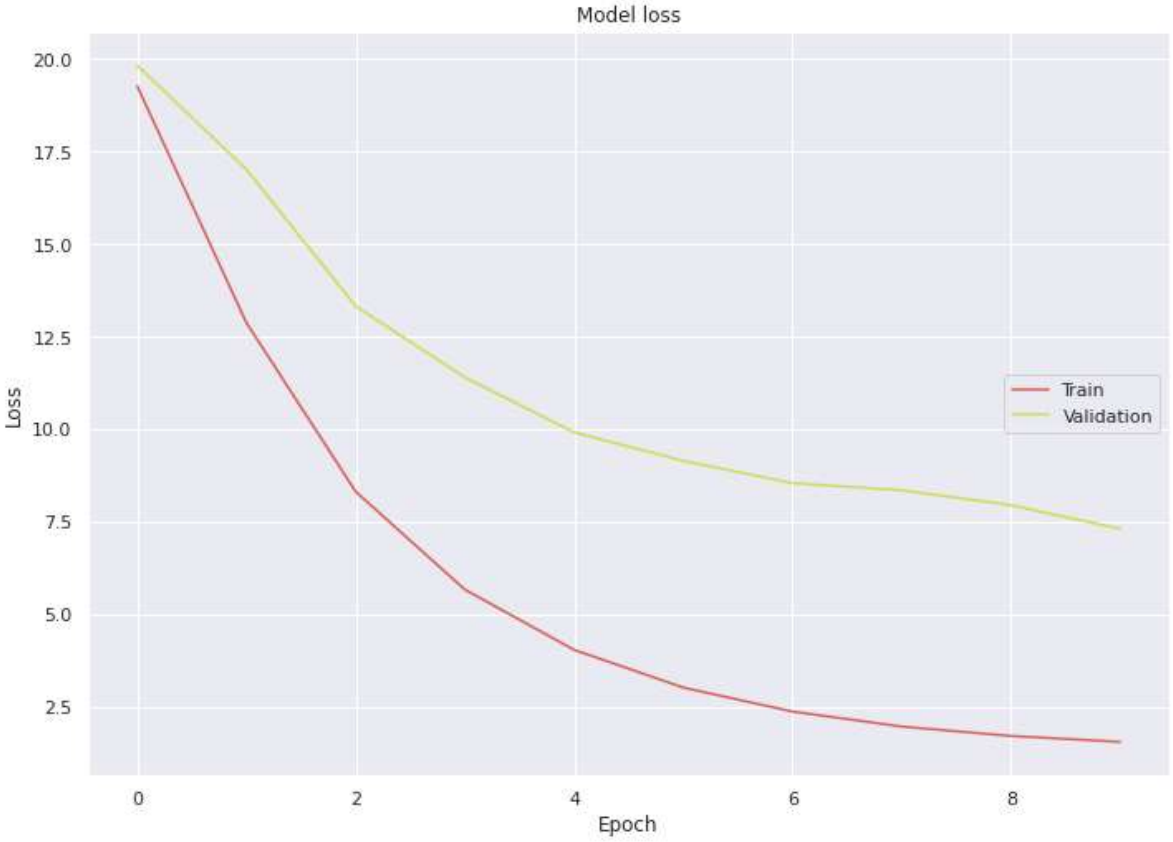
```
history_2 = model_2.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Epoch 1/10
11693/11693 [==============================] - 210s 18ms/step - loss: 19.271
4 - categorical_accuracy: 0.1146 - val_loss: 19.8313 - val_categorical_accur
acy: 0.0000e+00
Epoch 2/10
11693/11693 [==============================] - 210s 18ms/step - loss: 12.874
7 - categorical_accuracy: 0.1274 - val_loss: 17.0186 - val_categorical_accur
acy: 0.0000e+00
Epoch 3/10
11693/11693 [==============================] - 210s 18ms/step - loss: 8.3172
- categorical_accuracy: 0.3775 - val_loss: 13.3244 - val_categorical_accurac
y: 0.0000e+00
Epoch 4/10
11693/11693 [==============================] - 210s 18ms/step - loss: 5.6690
- categorical_accuracy: 0.4852 - val_loss: 11.4049 - val_categorical_accurac
y: 0.0000e+00
Epoch 5/10
11693/11693 [==============================] - 210s 18ms/step - loss: 4.0415
- categorical_accuracy: 0.5333 - val_loss: 9.9179 - val_categorical_accurac
y: 0.0000e+00
Epoch 6/10
11693/11693 [==============================] - 209s 18ms/step - loss: 3.0223
- categorical_accuracy: 0.5735 - val_loss: 9.1476 - val_categorical_accurac
y: 0.0000e+00
Epoch 7/10
11693/11693 [==============================] - 209s 18ms/step - loss: 2.3748
- categorical_accuracy: 0.6146 - val_loss: 8.5456 - val_categorical_accurac
y: 0.0000e+00
Epoch 8/10
11693/11693 [==============================] - 209s 18ms/step - loss: 1.9693
- categorical_accuracy: 0.6367 - val_loss: 8.3567 - val_categorical_accurac
y: 0.0000e+00
Epoch 9/10
11693/11693 [==============================] - 209s 18ms/step - loss: 1.7159
- categorical_accuracy: 0.6505 - val_loss: 7.9502 - val_categorical_accurac
y: 0.0000e+00
Epoch 10/10
11693/11693 [==============================] - 210s 18ms/step - loss: 1.5561
- categorical_accuracy: 0.6608 - val_loss: 7.3123 - val_categorical_accurac
y: 0.0000e+00
```

In [34]:

```
plt.plot(history_2.history['categorical_accuracy'])
plt.plot(history_2.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()
```





In [35]:

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

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

```
1444/1444 [==============================] - 8s 6ms/step - loss: 2.0421 - ca
tegorical_accuracy: 0.6249
Test loss, test accuracy: [2.042070150375366, 0.6248944401741028]
```

Регуляризация и сброс нейронов значительно помогли — модель показывает 62% 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?

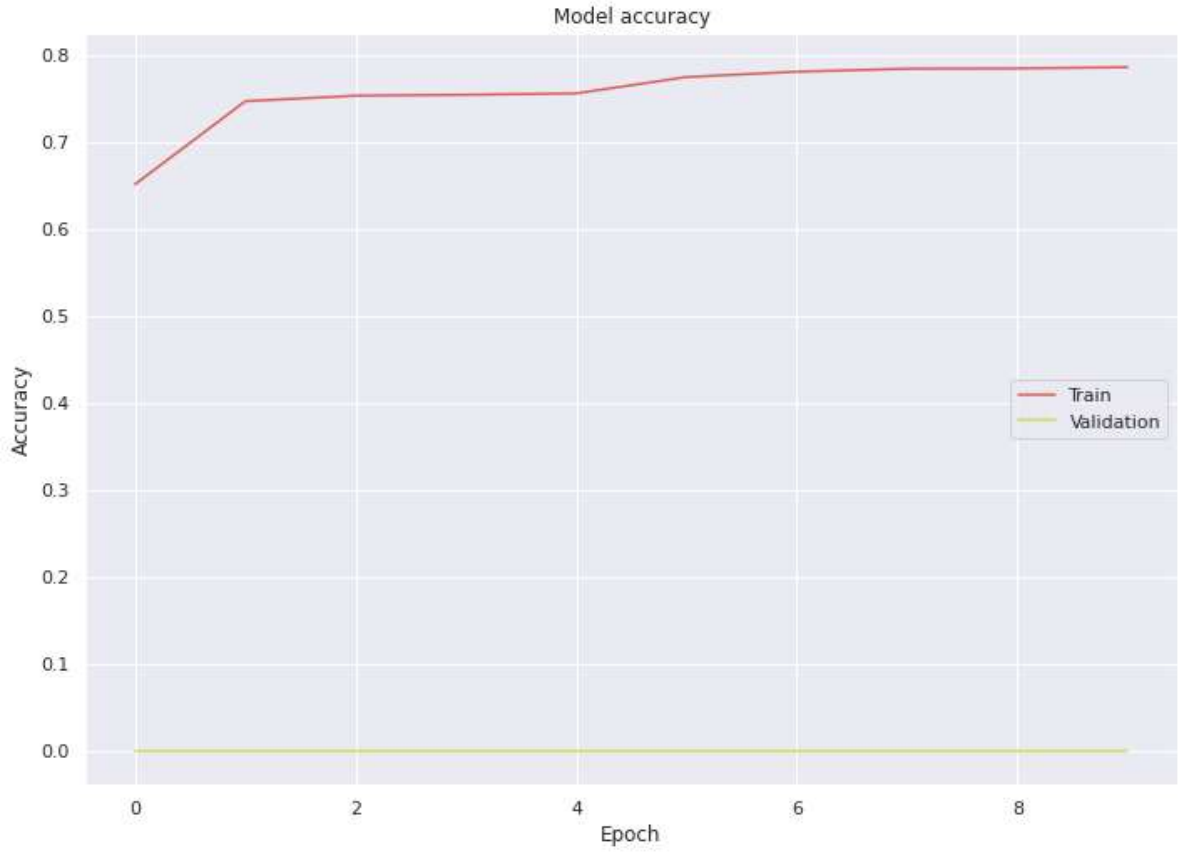
In [36]:

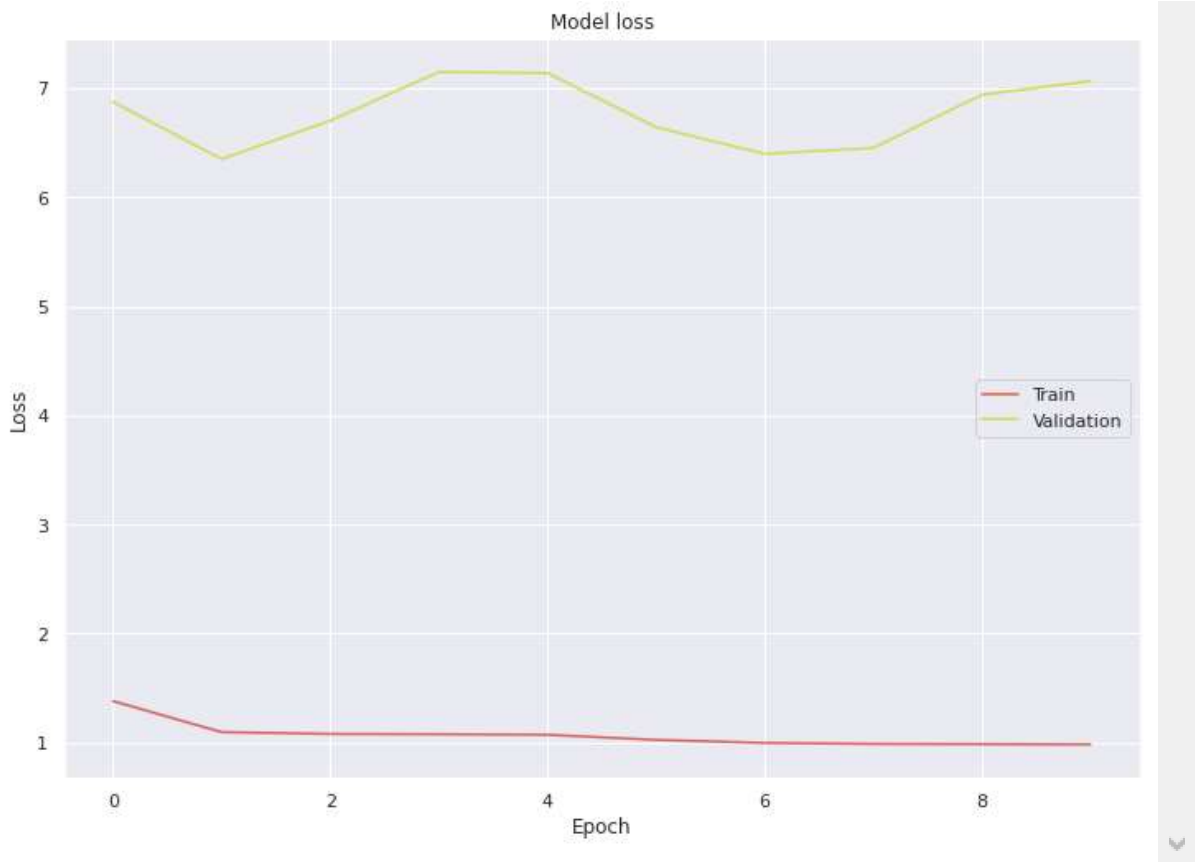
```
from tensorflow.keras.optimizers import SGD

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

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

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

```
Epoch 1/10
11693/11693 [==============================] - 237s 20ms/step - loss: 1.3790
- categorical_accuracy: 0.6511 - val_loss: 6.8754 - val_categorical_accurac
y: 0.0000e+00
Epoch 2/10
11693/11693 [==============================] - 237s 20ms/step - loss: 1.0942
- categorical_accuracy: 0.7466 - val_loss: 6.3502 - val_categorical_accurac
y: 0.0000e+00
Epoch 3/10
11693/11693 [==============================] - 237s 20ms/step - loss: 1.0778
- categorical_accuracy: 0.7529 - val_loss: 6.7014 - val_categorical_accurac
y: 0.0000e+00
Epoch 4/10
11693/11693 [==============================] - 237s 20ms/step - loss: 1.0749
- categorical_accuracy: 0.7540 - val_loss: 7.1483 - val_categorical_accurac
y: 0.0000e+00
Epoch 5/10
11693/11693 [==============================] - 237s 20ms/step - loss: 1.0698
- categorical_accuracy: 0.7556 - val_loss: 7.1391 - val_categorical_accurac
y: 0.0000e+00
Epoch 6/10
11693/11693 [==============================] - 237s 20ms/step - loss: 1.0231
- categorical_accuracy: 0.7744 - val_loss: 6.6427 - val_categorical_accurac
y: 0.0000e+00
Epoch 7/10
11693/11693 [==============================] - 237s 20ms/step - loss: 0.9952
- categorical_accuracy: 0.7805 - val_loss: 6.3981 - val_categorical_accurac
y: 0.0000e+00
Epoch 8/10
11693/11693 [==============================] - 237s 20ms/step - loss: 0.9859
- categorical_accuracy: 0.7839 - val_loss: 6.4524 - val_categorical_accurac
y: 0.0000e+00
Epoch 9/10
11693/11693 [==============================] - 237s 20ms/step - loss: 0.9838
- categorical_accuracy: 0.7843 - val_loss: 6.9396 - val_categorical_accurac
y: 0.0000e+00
Epoch 10/10
11693/11693 [==============================] - 237s 20ms/step - loss: 0.9809
- categorical_accuracy: 0.7857 - val_loss: 7.0662 - val_categorical_accurac
y: 0.0000e+00
```

In [38]:

```
plt.plot(history_3.history['categorical_accuracy'])
plt.plot(history_3.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

plt.plot(history_3.history['loss'])
plt.plot(history_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()
```





In [39]:

```
results_3 = model_2.evaluate(x_test, y_test)

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

```
1444/1444 [==============================] - 8s 6ms/step - loss: 1.5414 - ca
tegorical_accuracy: 0.7253
Test loss, test accuracy: [1.5414141416549683, 0.7252733111381531]
```

Динамически изменяемая скорость обучения улучшила результат — 72% 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 28×28 первых 10 букв латинского алфавита (*A_ ... _J*, соответственно). Обучающая выборка содержит порядка 500 тыс. изображений, а тестовая – около 19 тыс.

Данные можно скачать по ссылке:

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

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

Задание 1

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

In [1]:

```
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 [4]:

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

! pip show tensorflow-gpu
```

```
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: astunparse, opt-einsum, six, absl-py, tensorflow-estimator, termco
lor, grpcio, wrapt, h5py, gast, google-pasta, numpy, tensorboard, wheel, sci
py, keras-preprocessing, protobuf
Required-by:
```

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 [9]:

```
x = np.asarray(list(dataframe['data']))[..., np.newaxis]

x = tf.keras.utils.normalize(x, axis = 1)

x.shape
```

Out[9]:

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

In [10]:

```
x_test = np.asarray(list(dataframe_test['data']))[..., np.newaxis]

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

x_test.shape
```

Out[10]:

```
(46195, 28, 28, 1)
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

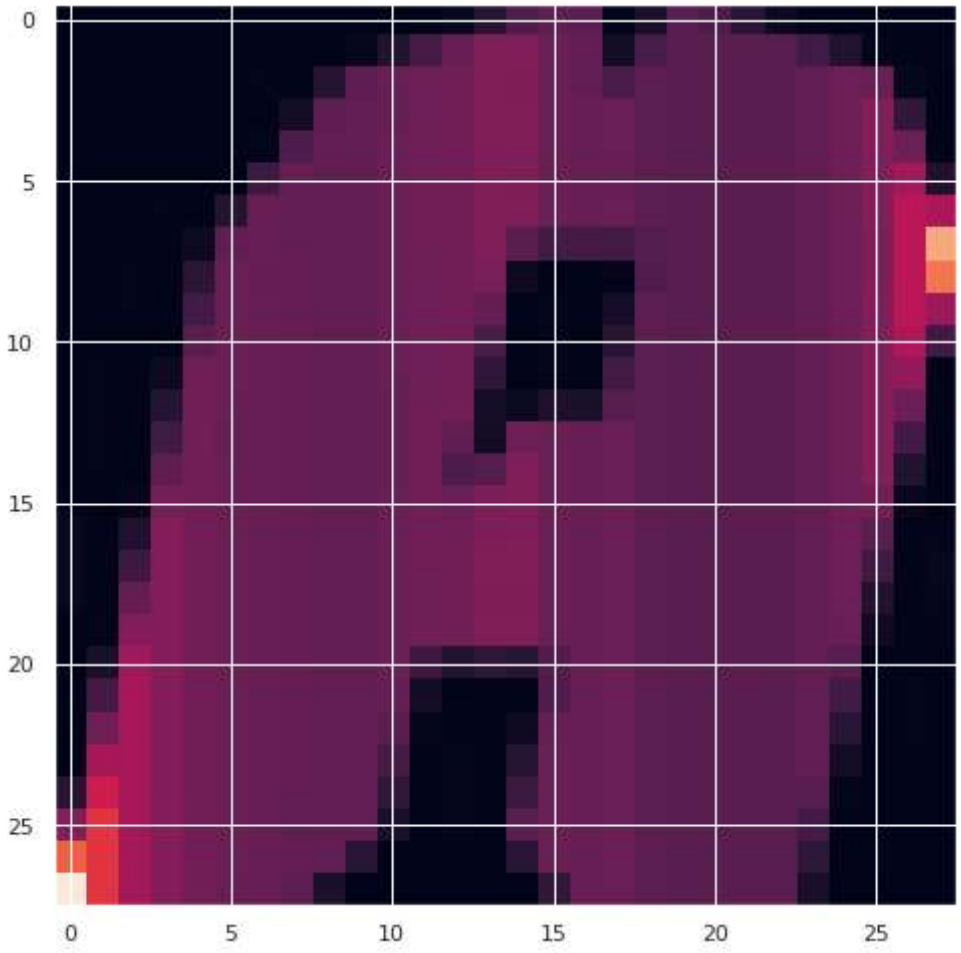
In [12]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

In [13]:

```
plt.imshow(x[100].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,
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/py
thon/ops/resource_variable_ops.py:1666: calling BaseResourceVariable.__init_
_ (from tensorflow.python.ops.resource_variable_ops) with constraint is depr
ecated 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
```

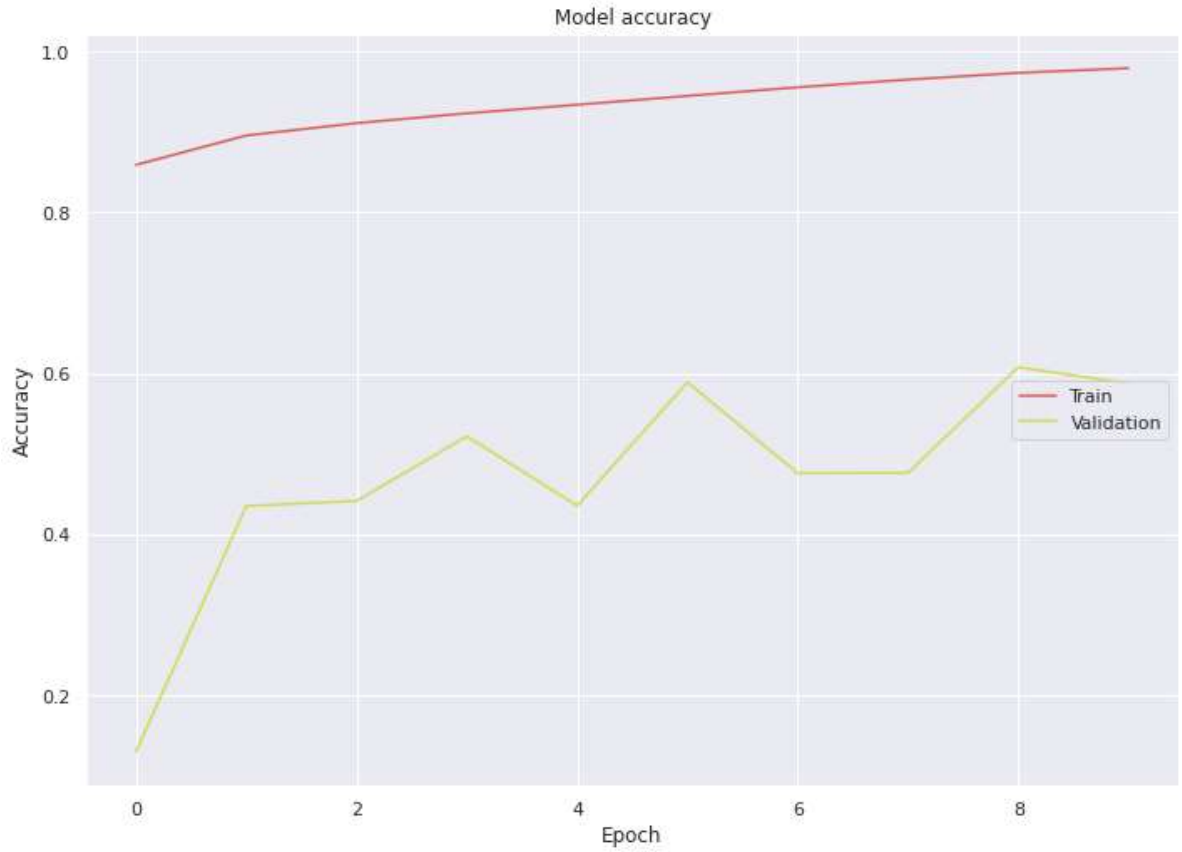
In [24]:

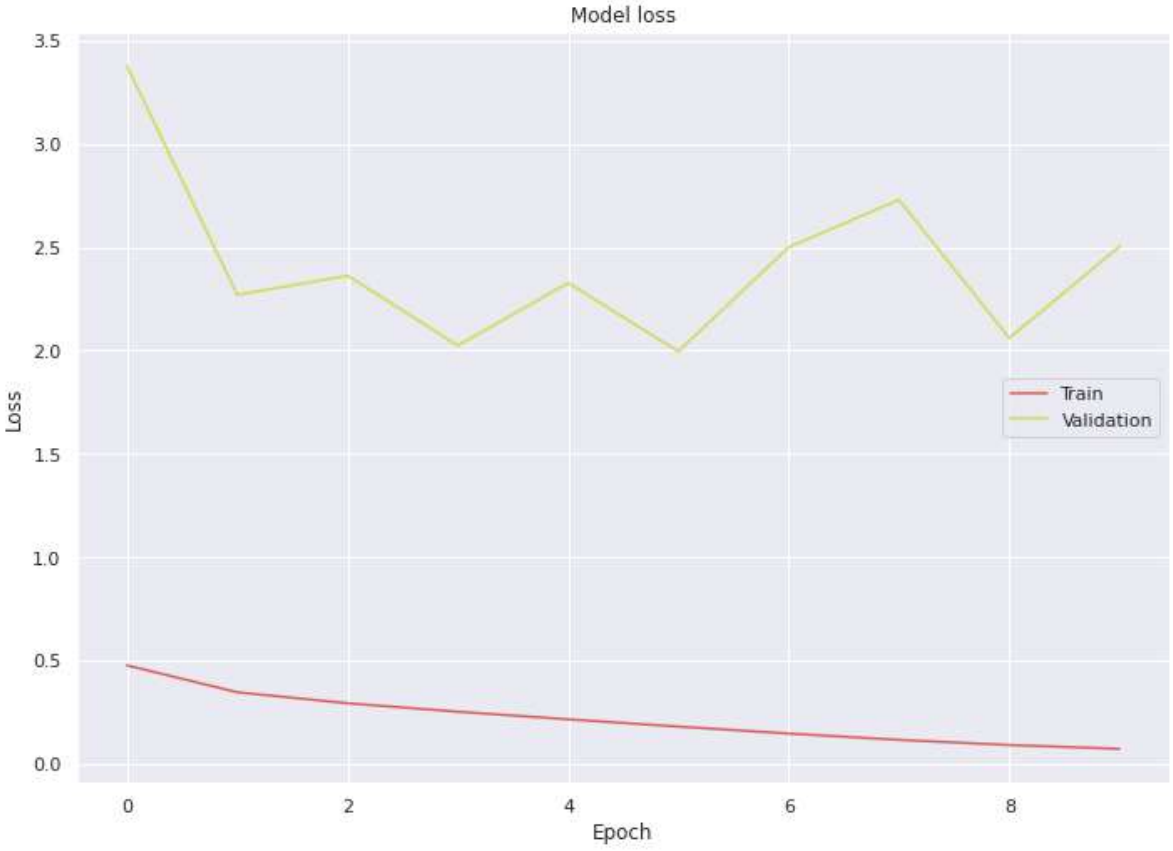
```
history = model.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Train on 374175 samples, validate on 41576 samples
Epoch 1/10
374175/374175 [==============================] - 120s 321us/sample - loss:
0.4768 - categorical_accuracy: 0.8591 - val_loss: 3.3758 - val_categorical_a
ccuracy: 0.1301
Epoch 2/10
374175/374175 [==============================] - 121s 324us/sample - loss:
0.3468 - categorical_accuracy: 0.8955 - val_loss: 2.2694 - val_categorical_a
ccuracy: 0.4353
Epoch 3/10
374175/374175 [==============================] - 121s 324us/sample - loss:
0.2936 - categorical_accuracy: 0.9108 - val_loss: 2.3619 - val_categorical_a
ccuracy: 0.4416
Epoch 4/10
374175/374175 [==============================] - 121s 322us/sample - loss:
0.2526 - categorical_accuracy: 0.9231 - val_loss: 2.0237 - val_categorical_a
ccuracy: 0.5219
Epoch 5/10
374175/374175 [==============================] - 121s 323us/sample - loss:
0.2159 - categorical_accuracy: 0.9337 - val_loss: 2.3267 - val_categorical_a
ccuracy: 0.4358
Epoch 6/10
374175/374175 [==============================] - 121s 322us/sample - loss:
0.1808 - categorical_accuracy: 0.9447 - val_loss: 1.9957 - val_categorical_a
ccuracy: 0.5889
Epoch 7/10
374175/374175 [==============================] - 120s 322us/sample - loss:
0.1470 - categorical_accuracy: 0.9554 - val_loss: 2.5000 - val_categorical_a
ccuracy: 0.4760
Epoch 8/10
374175/374175 [==============================] - 120s 322us/sample - loss:
0.1164 - categorical_accuracy: 0.9649 - val_loss: 2.7292 - val_categorical_a
ccuracy: 0.4765
Epoch 9/10
374175/374175 [==============================] - 120s 321us/sample - loss:
0.0915 - categorical_accuracy: 0.9733 - val_loss: 2.0604 - val_categorical_a
ccuracy: 0.6077
Epoch 10/10
374175/374175 [==============================] - 120s 321us/sample - loss:
0.0729 - categorical_accuracy: 0.9791 - val_loss: 2.5033 - val_categorical_a
ccuracy: 0.5875
```

In [25]:

```
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

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 [26]:

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

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

```
Test loss, test accuracy: [0.5405509961460728, 0.89239097]
```

Лучшая точность построенной модели на тестовой выборке составила 89%.

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
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_cross_from_logits,
                metrics = ['categorical_accuracy'])
```

In [29]:

```
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
_________________________________________________________________
```

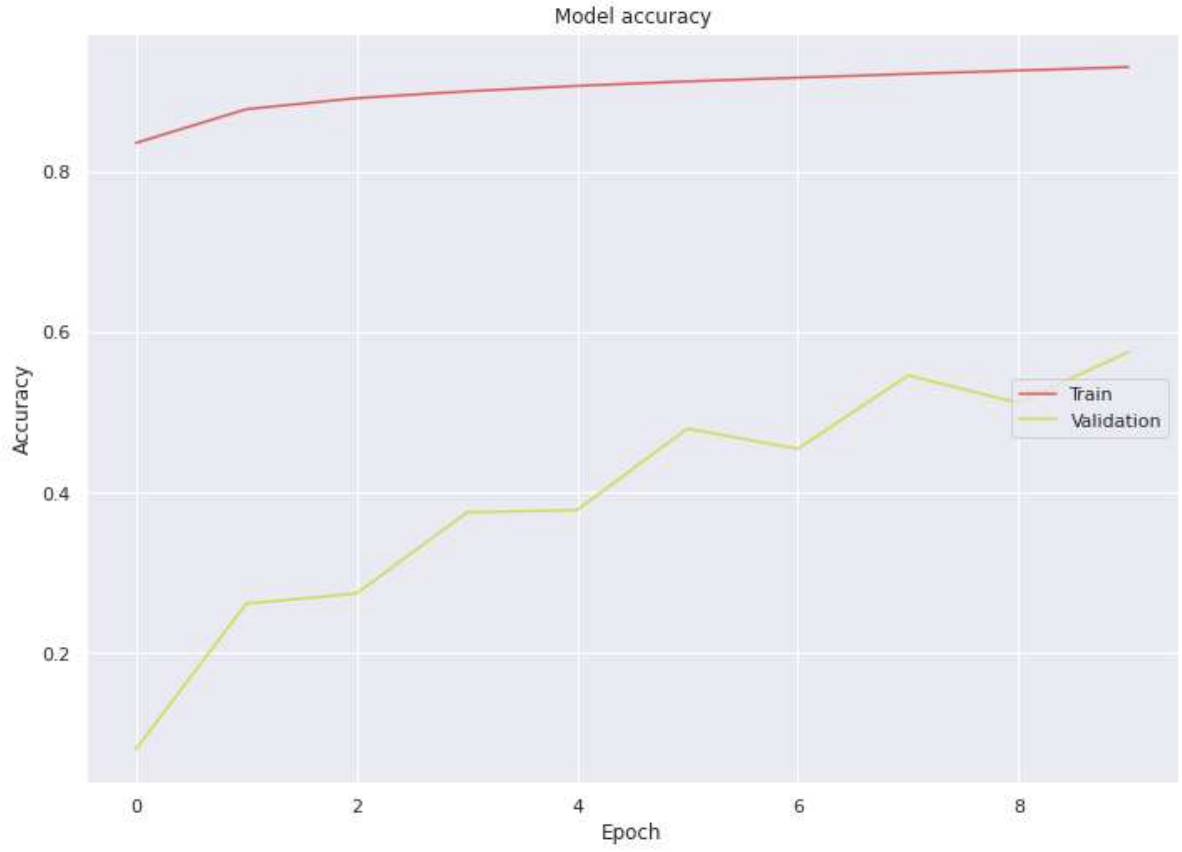
In [30]:

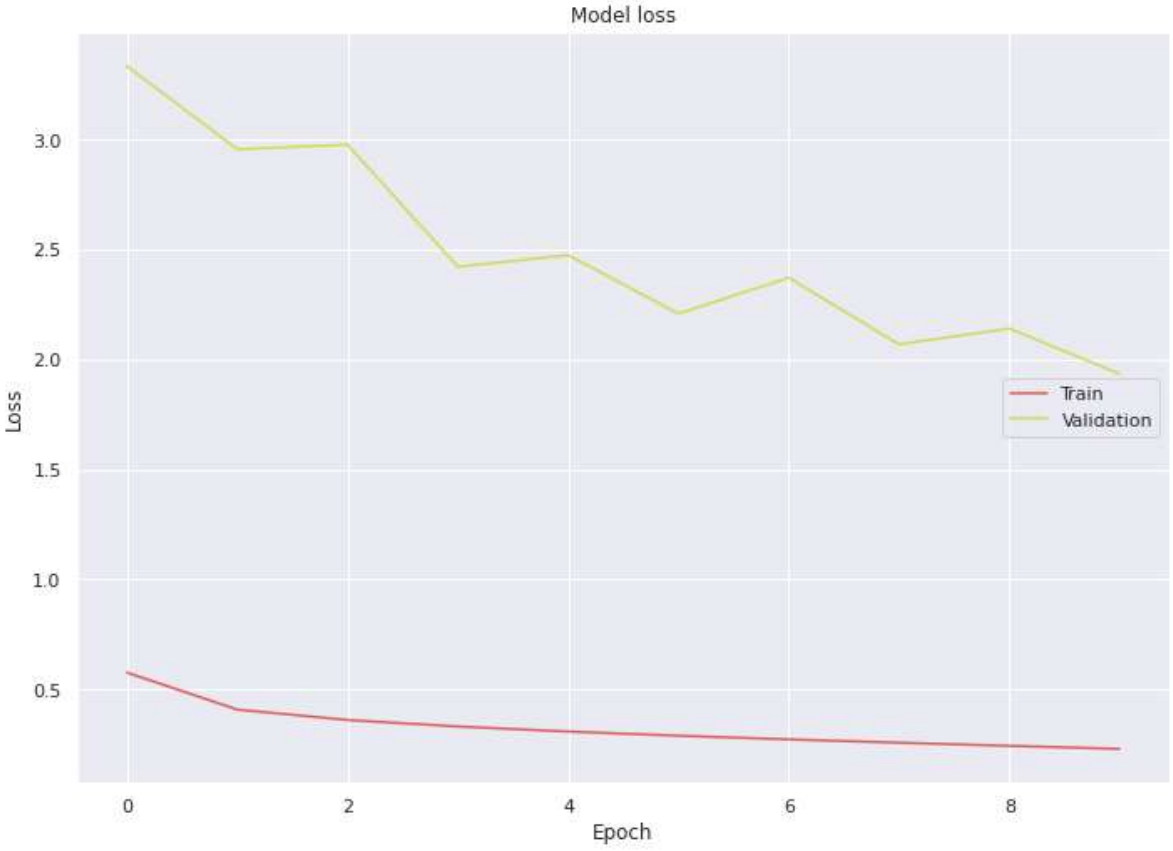
```
history_2 = model_2.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Train on 374175 samples, validate on 41576 samples
Epoch 1/10
374175/374175 [==============================] - 43s 114us/sample - loss: 0.
5757 - categorical_accuracy: 0.8356 - val_loss: 3.3338 - val_categorical_acc
uracy: 0.0800
Epoch 2/10
374175/374175 [==============================] - 42s 112us/sample - loss: 0.
4074 - categorical_accuracy: 0.8776 - val_loss: 2.9557 - val_categorical_acc
uracy: 0.2613
Epoch 3/10
374175/374175 [==============================] - 42s 112us/sample - loss: 0.
3602 - categorical_accuracy: 0.8914 - val_loss: 2.9767 - val_categorical_acc
uracy: 0.2739
Epoch 4/10
374175/374175 [==============================] - 42s 112us/sample - loss: 0.
3305 - categorical_accuracy: 0.8999 - val_loss: 2.4212 - val_categorical_acc
uracy: 0.3749
Epoch 5/10
374175/374175 [==============================] - 42s 113us/sample - loss: 0.
3076 - categorical_accuracy: 0.9067 - val_loss: 2.4744 - val_categorical_acc
uracy: 0.3780
Epoch 6/10
374175/374175 [==============================] - 42s 112us/sample - loss: 0.
2886 - categorical_accuracy: 0.9123 - val_loss: 2.2084 - val_categorical_acc
uracy: 0.4795
Epoch 7/10
374175/374175 [==============================] - 42s 111us/sample - loss: 0.
2720 - categorical_accuracy: 0.9170 - val_loss: 2.3712 - val_categorical_acc
uracy: 0.4542
Epoch 8/10
374175/374175 [==============================] - 43s 114us/sample - loss: 0.
2570 - categorical_accuracy: 0.9216 - val_loss: 2.0687 - val_categorical_acc
uracy: 0.5458
Epoch 9/10
374175/374175 [==============================] - 42s 112us/sample - loss: 0.
2426 - categorical_accuracy: 0.9259 - val_loss: 2.1407 - val_categorical_acc
uracy: 0.5115
Epoch 10/10
374175/374175 [==============================] - 42s 112us/sample - loss: 0.
2289 - categorical_accuracy: 0.9302 - val_loss: 1.9345 - val_categorical_acc
uracy: 0.5752
```

In [31]:

```
plt.plot(history_2.history['categorical_accuracy'])
plt.plot(history_2.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()
```





In [32]:

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

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

```
Test loss, test accuracy: [0.45766007790563945, 0.87918603]
```

Замена свёрточного слоя на операцию пулинга снизила точность на тестовой выборке до 87%.

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
                   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
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'])
```

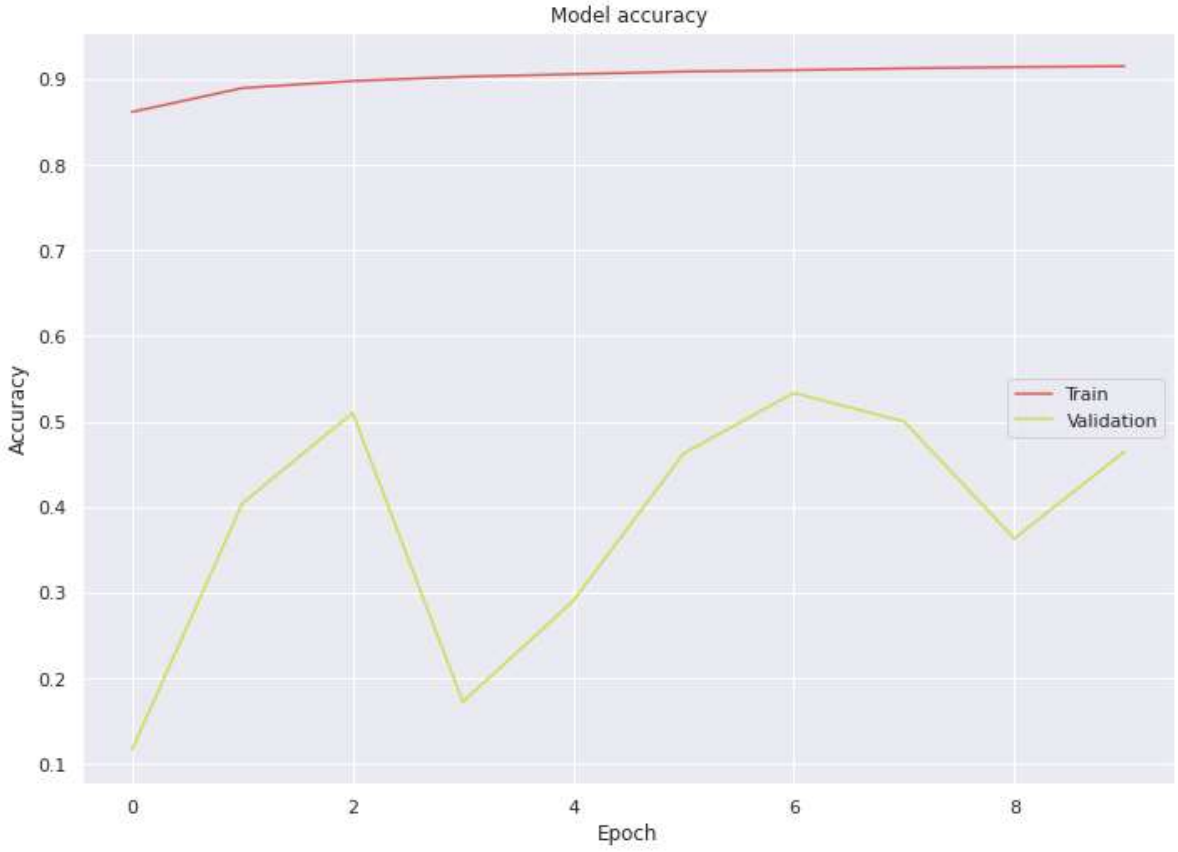
In [40]:

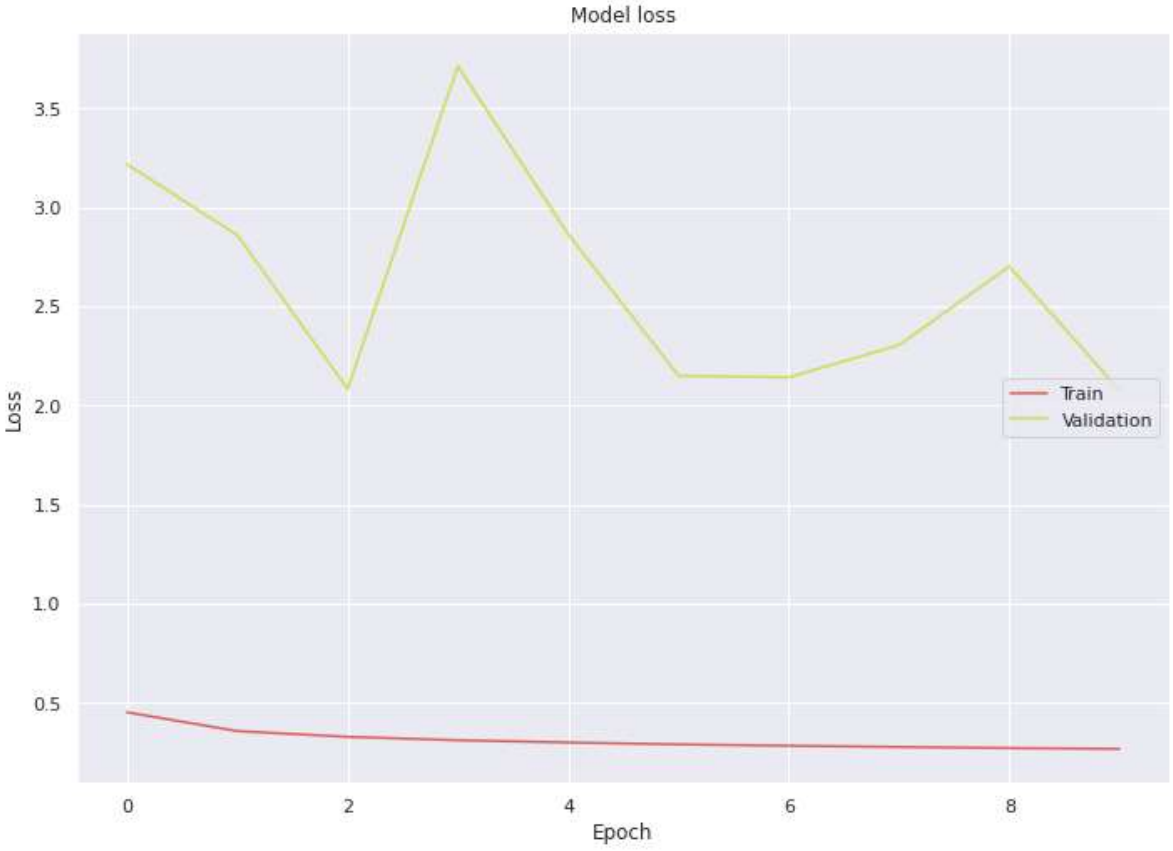
```
history_3 = model_3.fit(x = x, y = y, epochs = EPOCHS_N, validation_split = VAL_SPLIT_RATE)
```

```
Train on 374175 samples, validate on 41576 samples
Epoch 1/10
374175/374175 [==============================] - 39s 104us/sample - loss: 0.
4516 - categorical_accuracy: 0.8618 - val_loss: 3.2148 - val_categorical_acc
uracy: 0.1163
Epoch 2/10
374175/374175 [==============================] - 39s 103us/sample - loss: 0.
3568 - categorical_accuracy: 0.8898 - val_loss: 2.8585 - val_categorical_acc
uracy: 0.4042
Epoch 3/10
374175/374175 [==============================] - 39s 103us/sample - loss: 0.
3279 - categorical_accuracy: 0.8980 - val_loss: 2.0824 - val_categorical_acc
uracy: 0.5103
Epoch 4/10
374175/374175 [==============================] - 39s 103us/sample - loss: 0.
3107 - categorical_accuracy: 0.9030 - val_loss: 3.7105 - val_categorical_acc
uracy: 0.1722
Epoch 5/10
374175/374175 [==============================] - 39s 103us/sample - loss: 0.
2994 - categorical_accuracy: 0.9060 - val_loss: 2.8629 - val_categorical_acc
uracy: 0.2903
Epoch 6/10
374175/374175 [==============================] - 38s 103us/sample - loss: 0.
2897 - categorical_accuracy: 0.9091 - val_loss: 2.1490 - val_categorical_acc
uracy: 0.4624
Epoch 7/10
374175/374175 [==============================] - 39s 104us/sample - loss: 0.
2830 - categorical_accuracy: 0.9107 - val_loss: 2.1406 - val_categorical_acc
uracy: 0.5335
Epoch 8/10
374175/374175 [==============================] - 39s 105us/sample - loss: 0.
2765 - categorical_accuracy: 0.9128 - val_loss: 2.3037 - val_categorical_acc
uracy: 0.5000
Epoch 9/10
374175/374175 [==============================] - 38s 102us/sample - loss: 0.
2712 - categorical_accuracy: 0.9143 - val_loss: 2.7005 - val_categorical_acc
uracy: 0.3632
Epoch 10/10
374175/374175 [==============================] - 38s 103us/sample - loss: 0.
2672 - categorical_accuracy: 0.9153 - val_loss: 2.0758 - val_categorical_acc
uracy: 0.4649
```

In [41]:

```
plt.plot(history_3.history['categorical_accuracy'])
plt.plot(history_3.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

plt.plot(history_3.history['loss'])
plt.plot(history_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()
```





In [42]:

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

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

```
Test loss, test accuracy: [0.4807648109272324, 0.85974675]
```

Удивительно, но *LeNet-5* показала результат хуже, чем первая и вторая модель — 85% на тестовой выборке. Объяснить это можно тем, что первая модель содержит свёрточный слой с большей выходной размерностью.

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 — 62%;
        - с адаптивным шагом — 72%;
- модель с двумя свёрточными слоями и одним полносвязным — 89%;
- модель с одним свёрточным слоем, операцией пулинга и одним полносвязным — 87%;
- *LeNet-5* — два свёрточных слоя, две операции пулинга, два полносвязных слоя — 85%.

Объяснение превосходства свёрточных сетей над полносвязными — такая архитектура просто предназначена для работы с изображениями.

Лабораторная работа №4

Реализация приложения по распознаванию номеров домов

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

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

Задание 1

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

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

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

In [1]:

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

! pip show tensorflow-gpu
```

```
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: protobuf, wrapt, google-pasta, numpy, h5py, scipy, gast, opt-einsu
m, termcolor, grpcio, tensorflow-estimator, six, keras-preprocessing, wheel,
absl-py, astunparse, tensorboard
Required-by:
```

In [0]:

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

In [0]:

```
import numpy as np
```

In [0]:

```
from tensorflow.keras.datasets import mnist

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

In [0]:

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

In [0]:

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

In [7]:

```
from tensorflow.keras.utils import to_categorical

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

y_train.shape
```

Out[7]:

```
(60000, 10)
```

In [0]:

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

In [0]:

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

In [10]:

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

Out[10]:

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

In [0]:

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

model = tf.keras.Sequential()

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

In [0]:

```
# 'sparse_categorical_crossentropy' gave NAN loss

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

In [13]:

```
model.summary()
```

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

In [0]:

```
EPOCHS_N = 20
```

In [15]:

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

```
Epoch 1/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.2824 - ca
tegorical_accuracy: 0.9159 - val_loss: 0.1430 - val_categorical_accuracy: 0.
9600
Epoch 2/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.1236 - ca
tegorical_accuracy: 0.9623 - val_loss: 0.1033 - val_categorical_accuracy: 0.
9691
Epoch 3/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0867 - ca
tegorical_accuracy: 0.9732 - val_loss: 0.0829 - val_categorical_accuracy: 0.
9761
Epoch 4/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0663 - ca
tegorical_accuracy: 0.9793 - val_loss: 0.0835 - val_categorical_accuracy: 0.
9758
Epoch 5/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0523 - ca
tegorical_accuracy: 0.9831 - val_loss: 0.0771 - val_categorical_accuracy: 0.
9777
Epoch 6/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0435 - ca
tegorical_accuracy: 0.9863 - val_loss: 0.0744 - val_categorical_accuracy: 0.
9792
Epoch 7/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0345 - ca
tegorical_accuracy: 0.9887 - val_loss: 0.0747 - val_categorical_accuracy: 0.
9788
Epoch 8/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0290 - ca
tegorical_accuracy: 0.9904 - val_loss: 0.0756 - val_categorical_accuracy: 0.
9788
Epoch 9/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0239 - ca
tegorical_accuracy: 0.9925 - val_loss: 0.0663 - val_categorical_accuracy: 0.
9821
Epoch 10/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0203 - ca
tegorical_accuracy: 0.9933 - val_loss: 0.0686 - val_categorical_accuracy: 0.
9823
Epoch 11/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0181 - ca
tegorical_accuracy: 0.9940 - val_loss: 0.0749 - val_categorical_accuracy: 0.
9814
Epoch 12/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0186 - ca
tegorical_accuracy: 0.9939 - val_loss: 0.0731 - val_categorical_accuracy: 0.
9824
Epoch 13/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0142 - ca
tegorical_accuracy: 0.9950 - val_loss: 0.0780 - val_categorical_accuracy: 0.
9826
Epoch 14/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0151 - ca
tegorical_accuracy: 0.9952 - val_loss: 0.0671 - val_categorical_accuracy: 0.
9828
Epoch 15/20
```

```
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0126 - ca
tegorical_accuracy: 0.9956 - val_loss: 0.0740 - val_categorical_accuracy: 0.
9828
Epoch 16/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0114 - ca
tegorical_accuracy: 0.9963 - val_loss: 0.0834 - val_categorical_accuracy: 0.
9816
Epoch 17/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0101 - ca
tegorical_accuracy: 0.9966 - val_loss: 0.0711 - val_categorical_accuracy: 0.
9851
Epoch 18/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0112 - ca
tegorical_accuracy: 0.9964 - val_loss: 0.0771 - val_categorical_accuracy: 0.
9827
Epoch 19/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0112 - ca
tegorical_accuracy: 0.9962 - val_loss: 0.0887 - val_categorical_accuracy: 0.
9807
Epoch 20/20
1594/1594 [==============================] - 6s 4ms/step - loss: 0.0101 - ca
tegorical_accuracy: 0.9966 - val_loss: 0.0739 - val_categorical_accuracy: 0.
9842
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

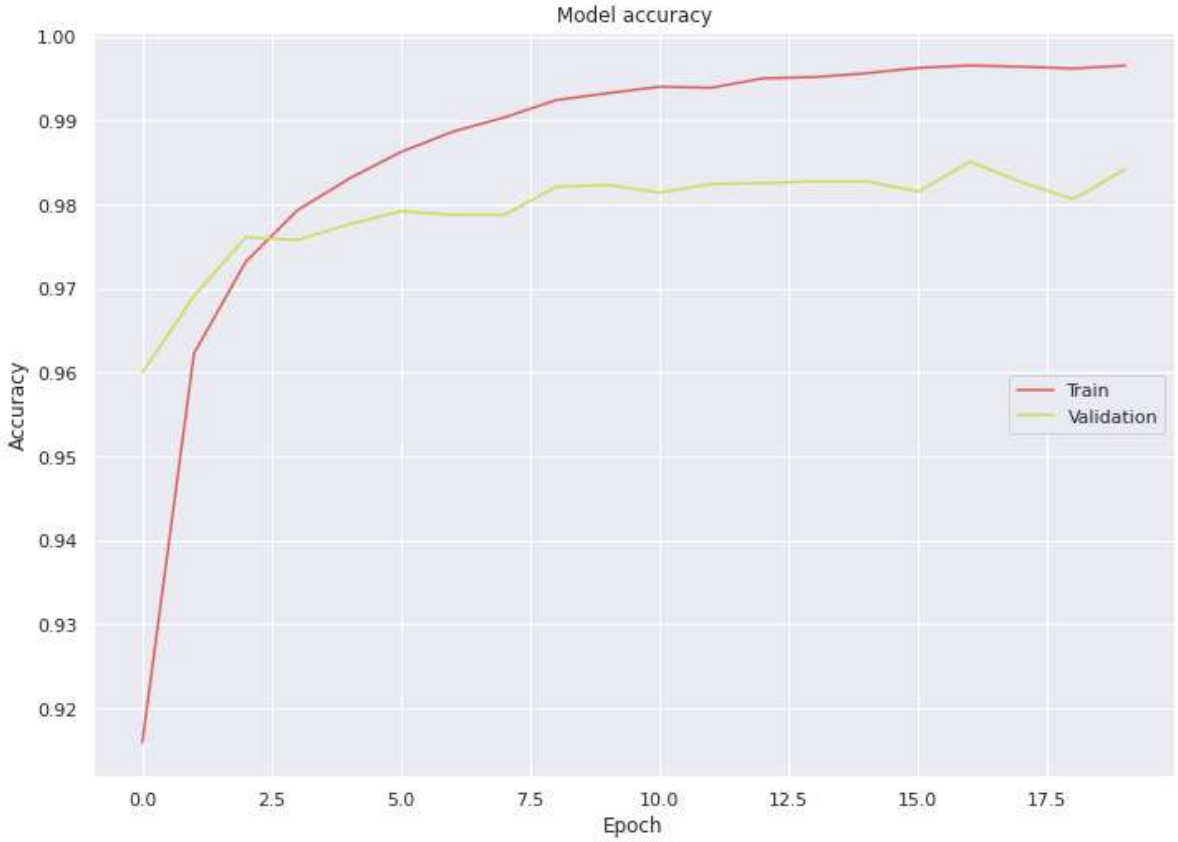
In [17]:

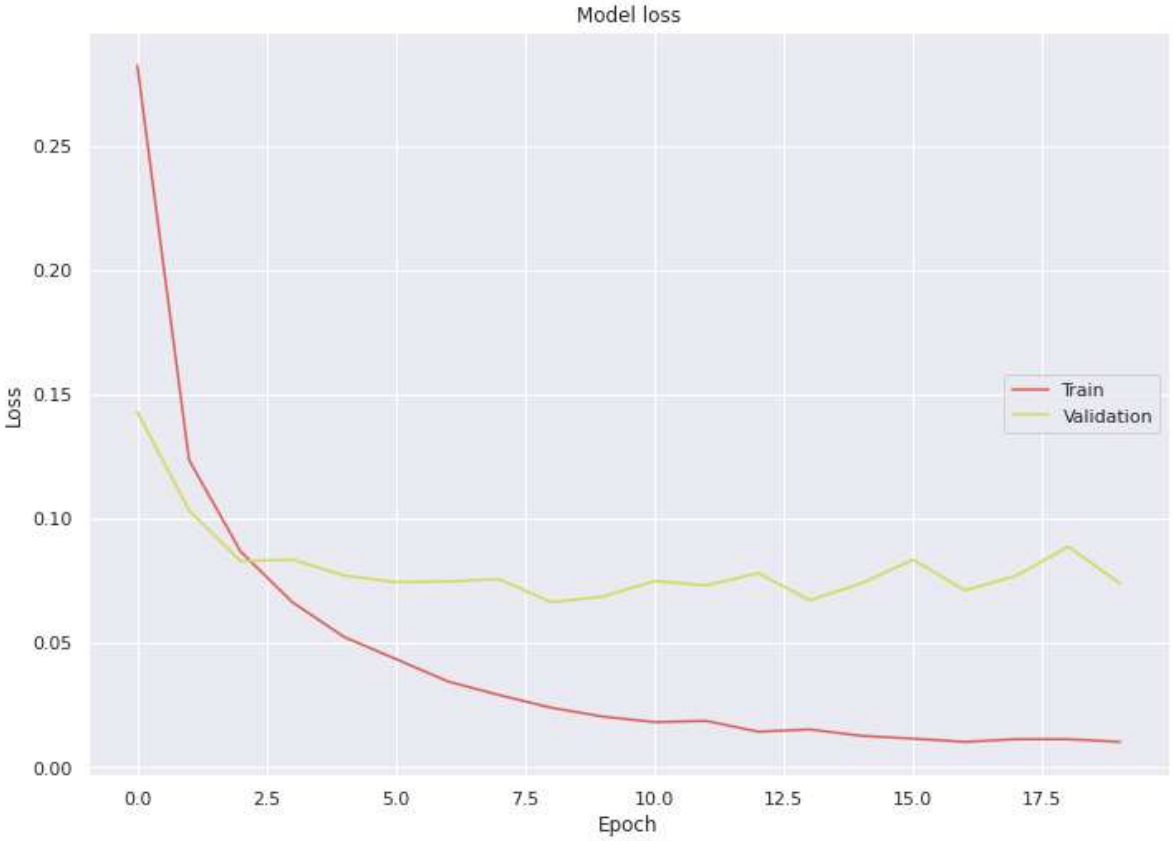
```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

In [18]:

```
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

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 [19]:

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

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

```
313/313 [==============================] - 1s 2ms/step - loss: 0.0597 - cate
gorical_accuracy: 0.9840
Test loss, test accuracy: [0.059687741100788116, 0.984000027179718]
```

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?

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 [24]:

```
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]:

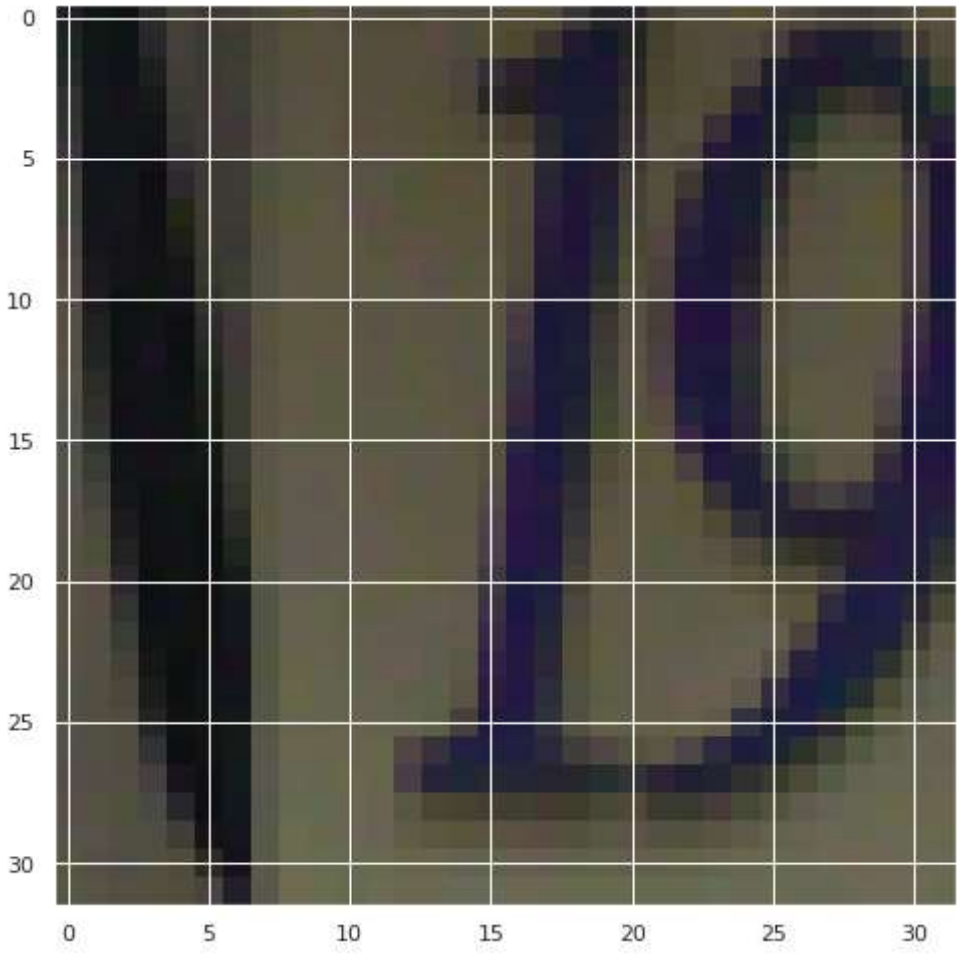
```
%matplotlib inline

import matplotlib.pyplot as plt
```

In [0]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

In [27]:

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

plt.show()
```



In [0]:

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

In [0]:

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

In [0]:

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

In [0]:

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

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

In [0]:

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

In [33]:

```
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 [34]:

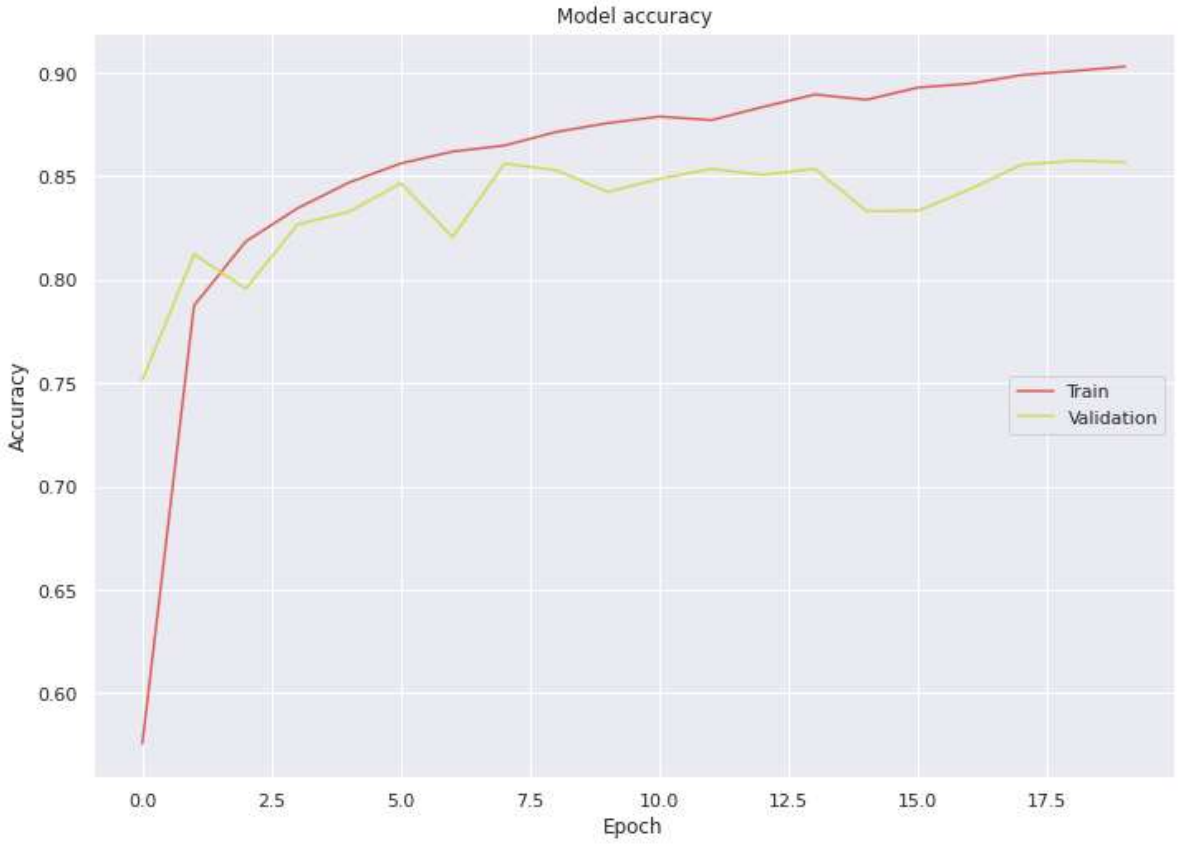
```
history_2 = model_2.fit(x = X_second_ds_train, y = y_second_ds_train_cat, validation_split
```

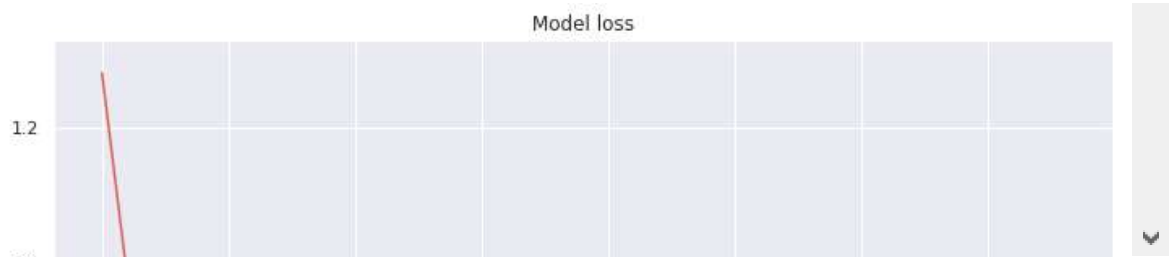
```
Epoch 1/20
1946/1946 [==============================] - 7s 4ms/step - loss: 1.2807 - ca
tegorical_accuracy: 0.5756 - val_loss: 0.7925 - val_categorical_accuracy: 0.
7516
Epoch 2/20
1946/1946 [==============================] - 8s 4ms/step - loss: 0.6788 - ca
tegorical_accuracy: 0.7875 - val_loss: 0.6073 - val_categorical_accuracy: 0.
8124
Epoch 3/20
1946/1946 [==============================] - 8s 4ms/step - loss: 0.5796 - ca
tegorical_accuracy: 0.8185 - val_loss: 0.6478 - val_categorical_accuracy: 0.
7957
Epoch 4/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.5294 - ca
tegorical_accuracy: 0.8346 - val_loss: 0.5545 - val_categorical_accuracy: 0.
8267
Epoch 5/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.4868 - ca
tegorical_accuracy: 0.8471 - val_loss: 0.5522 - val_categorical_accuracy: 0.
8330
Epoch 6/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.4613 - ca
tegorical_accuracy: 0.8563 - val_loss: 0.4898 - val_categorical_accuracy: 0.
8467
Epoch 7/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.4401 - ca
tegorical_accuracy: 0.8620 - val_loss: 0.5859 - val_categorical_accuracy: 0.
8206
Epoch 8/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.4305 - ca
tegorical_accuracy: 0.8649 - val_loss: 0.4773 - val_categorical_accuracy: 0.
8562
Epoch 9/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.4106 - ca
tegorical_accuracy: 0.8715 - val_loss: 0.4831 - val_categorical_accuracy: 0.
8530
Epoch 10/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3969 - ca
tegorical_accuracy: 0.8758 - val_loss: 0.5368 - val_categorical_accuracy: 0.
8425
Epoch 11/20
1946/1946 [==============================] - 8s 4ms/step - loss: 0.3839 - ca
tegorical_accuracy: 0.8790 - val_loss: 0.5044 - val_categorical_accuracy: 0.
8488
Epoch 12/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3895 - ca
tegorical_accuracy: 0.8772 - val_loss: 0.4950 - val_categorical_accuracy: 0.
8537
Epoch 13/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3666 - ca
tegorical_accuracy: 0.8836 - val_loss: 0.4976 - val_categorical_accuracy: 0.
8508
Epoch 14/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3539 - ca
tegorical_accuracy: 0.8896 - val_loss: 0.4870 - val_categorical_accuracy: 0.
8537
Epoch 15/20
```

```
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3575 - ca
tegorical_accuracy: 0.8871 - val_loss: 0.5541 - val_categorical_accuracy: 0.
8333
Epoch 16/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3422 - ca
tegorical_accuracy: 0.8930 - val_loss: 0.5480 - val_categorical_accuracy: 0.
8334
Epoch 17/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3316 - ca
tegorical_accuracy: 0.8949 - val_loss: 0.5171 - val_categorical_accuracy: 0.
8438
Epoch 18/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3165 - ca
tegorical_accuracy: 0.8990 - val_loss: 0.5014 - val_categorical_accuracy: 0.
8557
Epoch 19/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3111 - ca
tegorical_accuracy: 0.9010 - val_loss: 0.4844 - val_categorical_accuracy: 0.
8576
Epoch 20/20
1946/1946 [==============================] - 7s 4ms/step - loss: 0.3063 - ca
tegorical_accuracy: 0.9031 - val_loss: 0.4942 - val_categorical_accuracy: 0.
8569
```

In [35]:

```
plt.plot(history_2.history['categorical_accuracy'])
plt.plot(history_2.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()

plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'right')
plt.show()
```





In [36]:

```
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: 0.5545 - cate
gorical_accuracy: 0.8396
Test loss, test accuracy: [0.5544997453689575, 0.8396204710006714]
```

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

Эти данные более сложны для распознавания, что повлияло на результат — точность распознавания на тестовой выборке составила 83%.

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

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]:

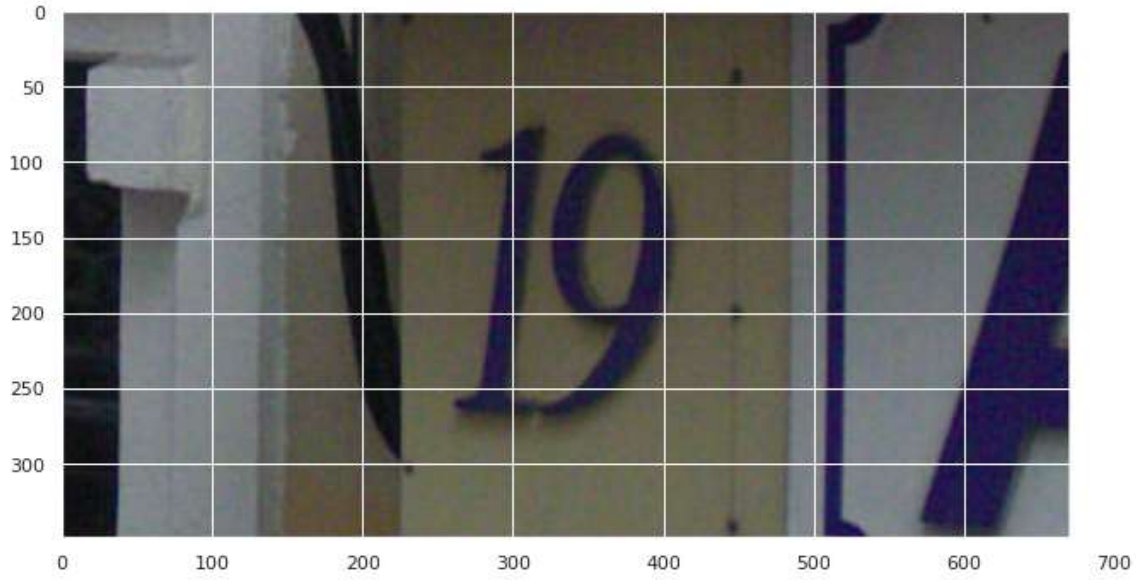
```
mat_to_pickle(os.path.join(first_ds_train_subdir, 'digitStruct.mat'), 'train_bbox')
mat_to_pickle(os.path.join(first_ds_test_subdir, 'digitStruct.mat'), 'test_bbox')
```

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 [45]:

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

plt.show()
```



In [46]:

```
train_bbox_data['label'][0]
```

Out[46]:

```
[1, 9]
```

Задание 3

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

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 [1]:

```
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
```

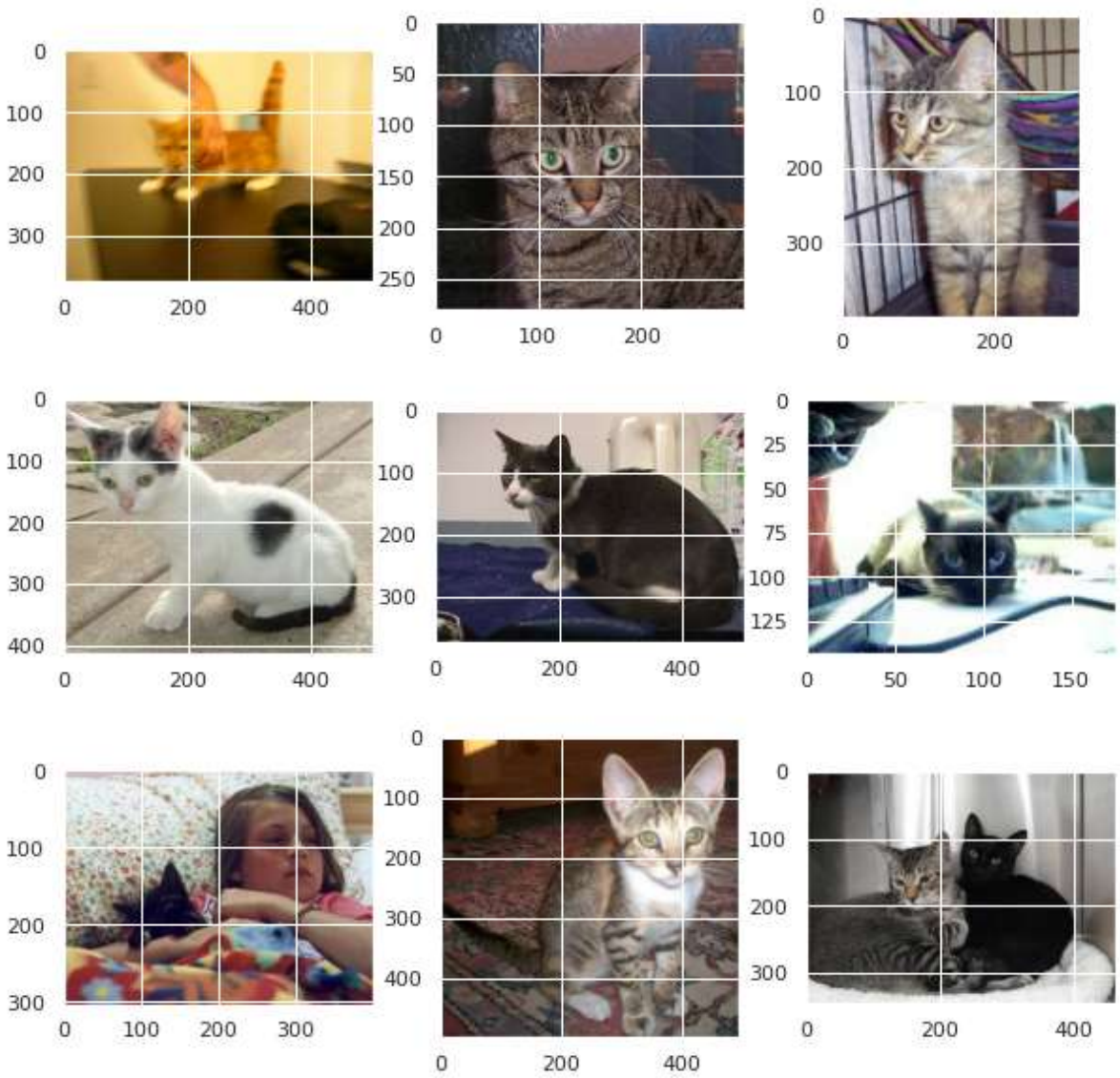
In [6]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

In [7]:

```
from matplotlib import pyplot
from matplotlib.image import imread

pyplot.rcParams["figure.figsize"] = (10, 10)

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

for i in range(9):

    pyplot.subplot(330 + 1 + i)

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

    pyplot.imshow(image_)

pyplot.show()
```



Изображения необходимо прирвести 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

        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 [10]:

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

! pip show tensorflow-gpu
```

```
     |████████████████████████████████| 516.2MB 22kB/s
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: astunparse, numpy, protobuf, h5py, gast, google-pasta, termcolor,
 tensorboard, opt-einsum, grpcio, six, wrapt, absl-py, wheel, scipy, keras-p
reprocessing, tensorflow-estimator
Required-by:
```

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
```

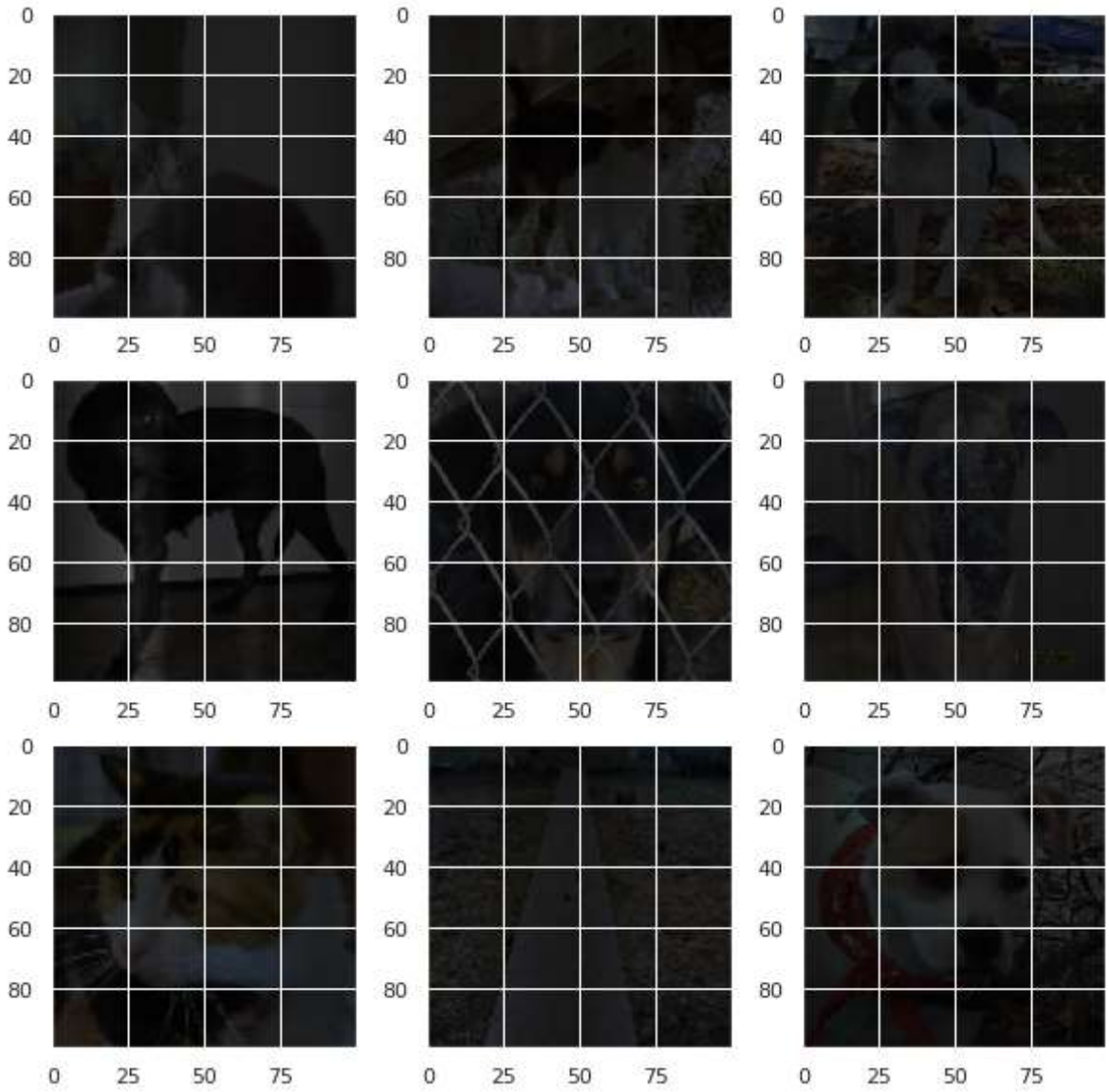
In [15]:

```
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):

    pyplot.subplot(330 + 1 + i)

    pyplot.imshow(X[i])

pyplot.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_WID
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 (MaxPooling2 (None, 25, 25, 32)        0
_________________________________________________________________
conv2d_2 (Conv2D)            (None, 25, 25, 64)        18496
_________________________________________________________________
max_pooling2d_2 (MaxPooling2 (None, 12, 12, 64)        0
_________________________________________________________________
flatten (Flatten)            (None, 9216)              0
_________________________________________________________________
dense (Dense)                (None, 512)               4719104
_________________________________________________________________
dense_1 (Dense)              (None, 1)                 513
=================================================================
Total params: 4,743,201
Trainable params: 4,743,201
Non-trainable params: 0
_________________________________________________________________
```

In [19]:

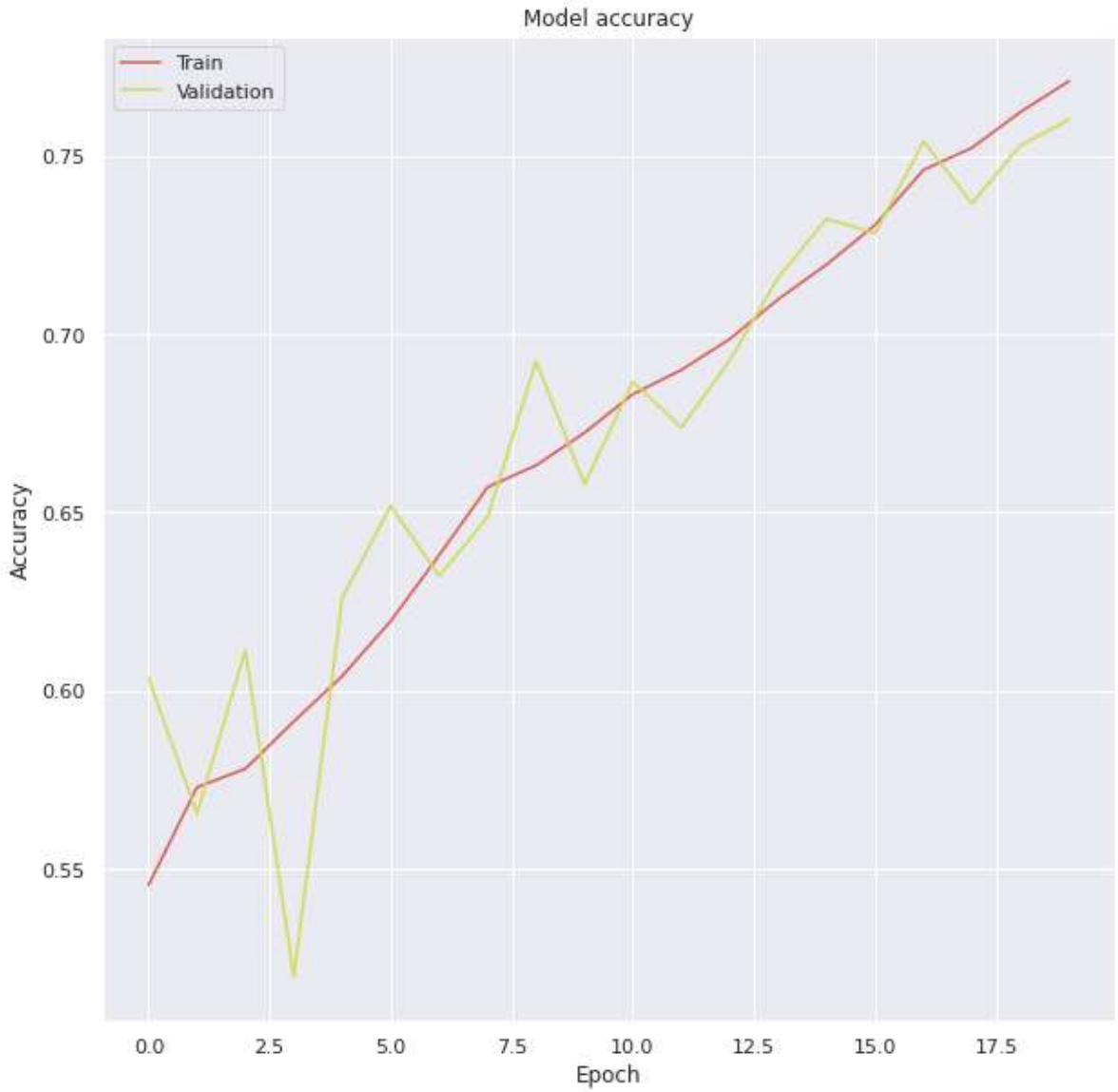
```
history = model.fit(x = X, y = y, epochs = 20, validation_split = 0.15)
```

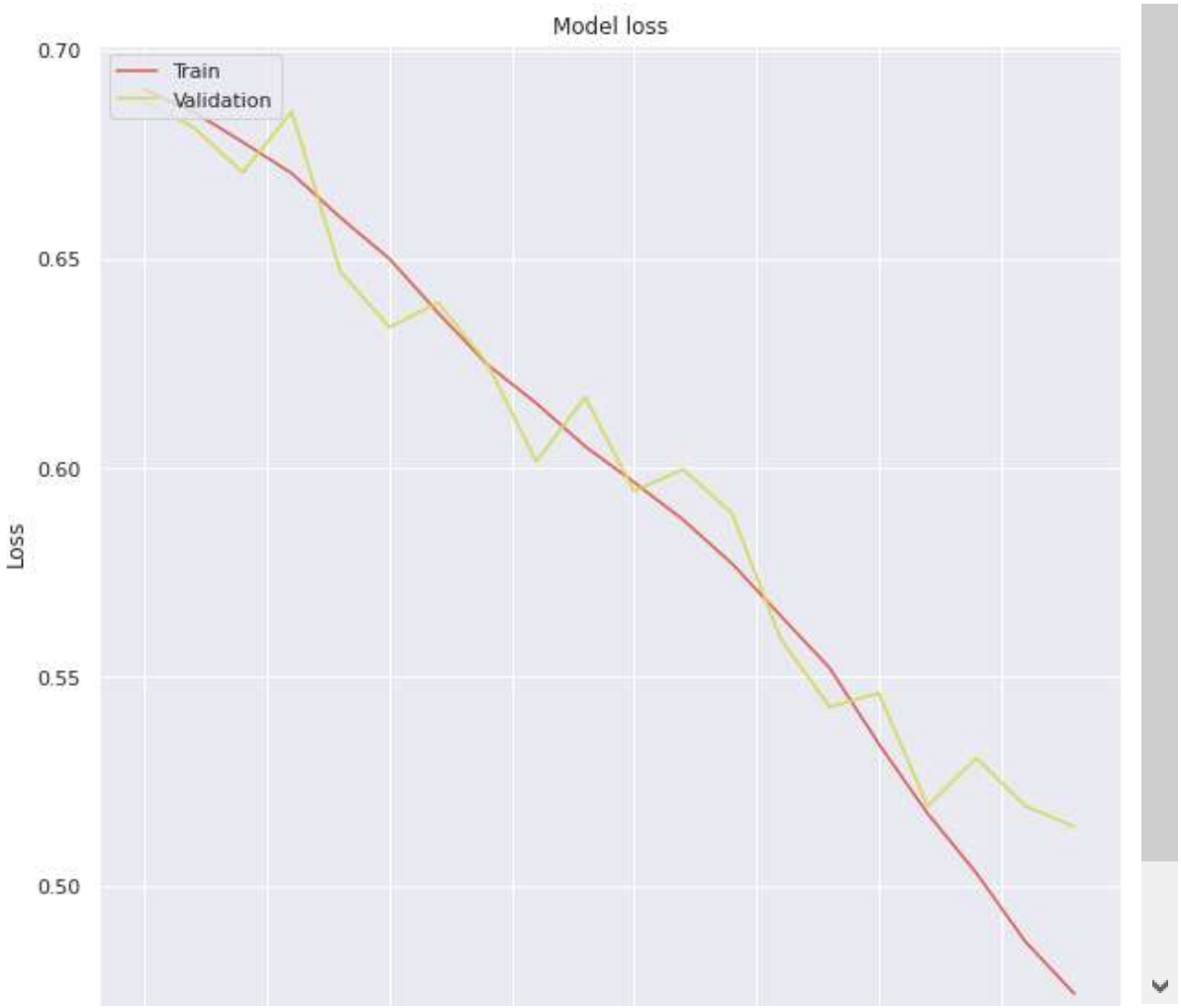
```
Epoch 1/20
611/611 [==============================] - 5s 8ms/step - loss: 0.6904 - accu
racy: 0.5454 - val_loss: 0.6876 - val_accuracy: 0.6035
Epoch 2/20
611/611 [==============================] - 5s 7ms/step - loss: 0.6851 - accu
racy: 0.5728 - val_loss: 0.6814 - val_accuracy: 0.5655
Epoch 3/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6779 - accu
racy: 0.5781 - val_loss: 0.6706 - val_accuracy: 0.6113
Epoch 4/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6704 - accu
racy: 0.5914 - val_loss: 0.6852 - val_accuracy: 0.5197
Epoch 5/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6598 - accu
racy: 0.6041 - val_loss: 0.6469 - val_accuracy: 0.6261
Epoch 6/20
611/611 [==============================] - 4s 7ms/step - loss: 0.6500 - accu
racy: 0.6195 - val_loss: 0.6336 - val_accuracy: 0.6519
Epoch 7/20
611/611 [==============================] - 5s 7ms/step - loss: 0.6370 - accu
racy: 0.6380 - val_loss: 0.6395 - val_accuracy: 0.6322
Epoch 8/20
611/611 [==============================] - 5s 7ms/step - loss: 0.6247 - accu
racy: 0.6571 - val_loss: 0.6249 - val_accuracy: 0.6487
Epoch 9/20
611/611 [==============================] - 5s 7ms/step - loss: 0.6155 - accu
racy: 0.6632 - val_loss: 0.6015 - val_accuracy: 0.6925
Epoch 10/20
611/611 [==============================] - 5s 7ms/step - loss: 0.6051 - accu
racy: 0.6724 - val_loss: 0.6168 - val_accuracy: 0.6580
Epoch 11/20
611/611 [==============================] - 5s 7ms/step - loss: 0.5965 - accu
racy: 0.6832 - val_loss: 0.5944 - val_accuracy: 0.6867
Epoch 12/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5876 - accu
racy: 0.6900 - val_loss: 0.5996 - val_accuracy: 0.6739
Epoch 13/20
611/611 [==============================] - 5s 7ms/step - loss: 0.5771 - accu
racy: 0.6987 - val_loss: 0.5891 - val_accuracy: 0.6928
Epoch 14/20
611/611 [==============================] - 5s 7ms/step - loss: 0.5645 - accu
racy: 0.7099 - val_loss: 0.5590 - val_accuracy: 0.7159
Epoch 15/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5521 - accu
racy: 0.7196 - val_loss: 0.5428 - val_accuracy: 0.7325
Epoch 16/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5340 - accu
racy: 0.7307 - val_loss: 0.5461 - val_accuracy: 0.7284
Epoch 17/20
611/611 [==============================] - 5s 8ms/step - loss: 0.5173 - accu
racy: 0.7462 - val_loss: 0.5191 - val_accuracy: 0.7542
Epoch 18/20
611/611 [==============================] - 5s 8ms/step - loss: 0.5030 - accu
racy: 0.7524 - val_loss: 0.5306 - val_accuracy: 0.7368
Epoch 19/20
611/611 [==============================] - 5s 8ms/step - loss: 0.4867 - accu
racy: 0.7625 - val_loss: 0.5191 - val_accuracy: 0.7530
```

```
Epoch 20/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4742 - accu
racy: 0.7711 - val_loss: 0.5142 - val_accuracy: 0.7603
```

In [20]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

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 = 'upper left')
plt.show()
```





In [21]:

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

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

```
63/63 [==============================] - 0s 4ms/step - loss: 0.5067 - accura
cy: 0.7615
Test loss, test accuracy: [0.5066752433776855, 0.7615000009536743]
```

Результат — 76% 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 + 4
  image = tf.image.random_crop(image, size = [NEW_IMAGE_WIDTH, NEW_IMAGE_WIDTH, 3])

  return image.numpy()
```

In [23]:

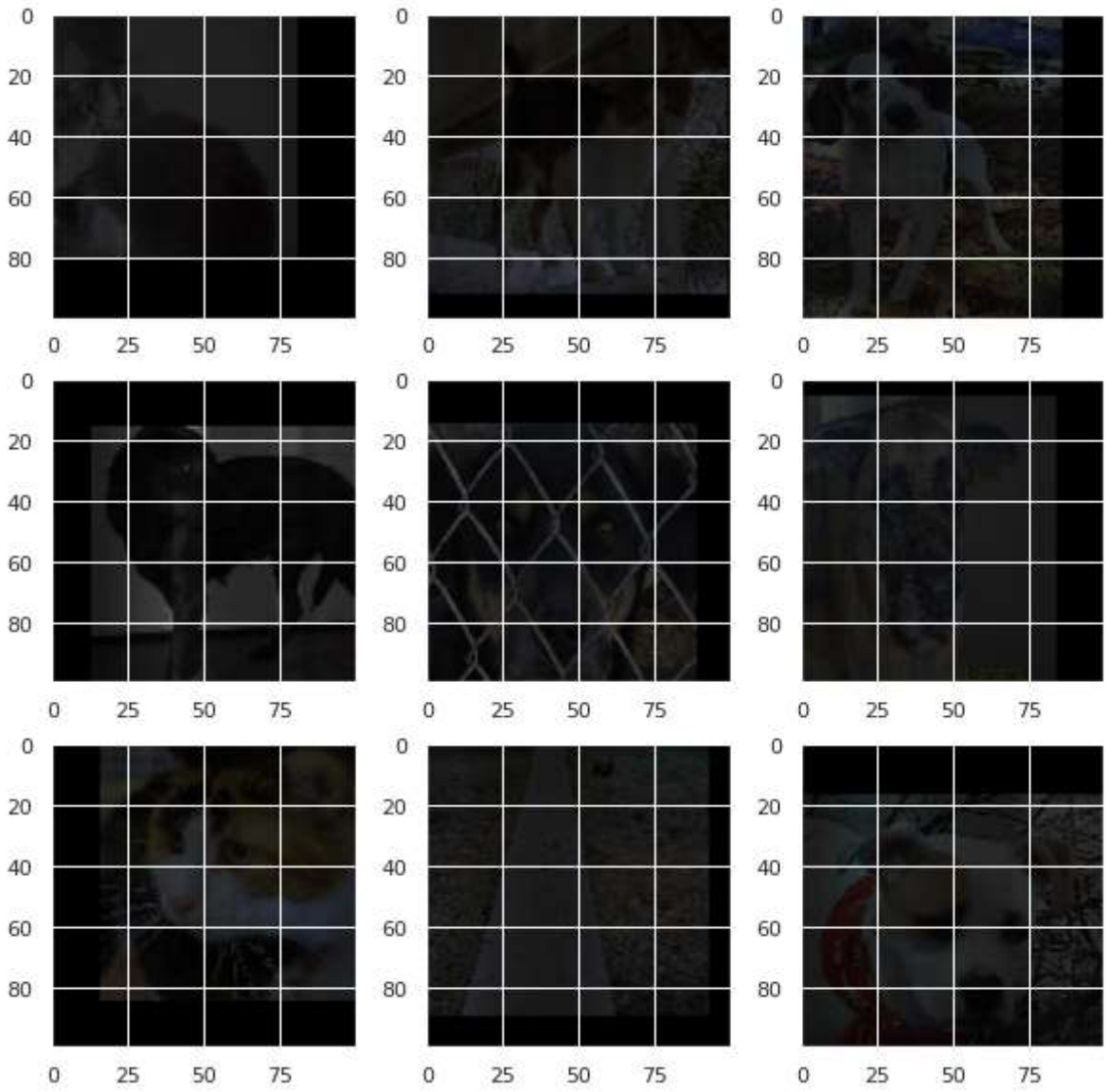
```
X_augmented = np.zeros_like(X)

for i, img in enumerate(X):

    X_augmented[i] = augment_image(img)

X_augmented.shape
```

Out[23]:

```
(23000, 100, 100, 3)
```

In [24]:

```
for i in range(9):

    pyplot.subplot(330 + 1 + i)

    pyplot.imshow(X_augmented[i])

pyplot.show()
```



In [0]:

```
y_augmented = y
```

In [26]:

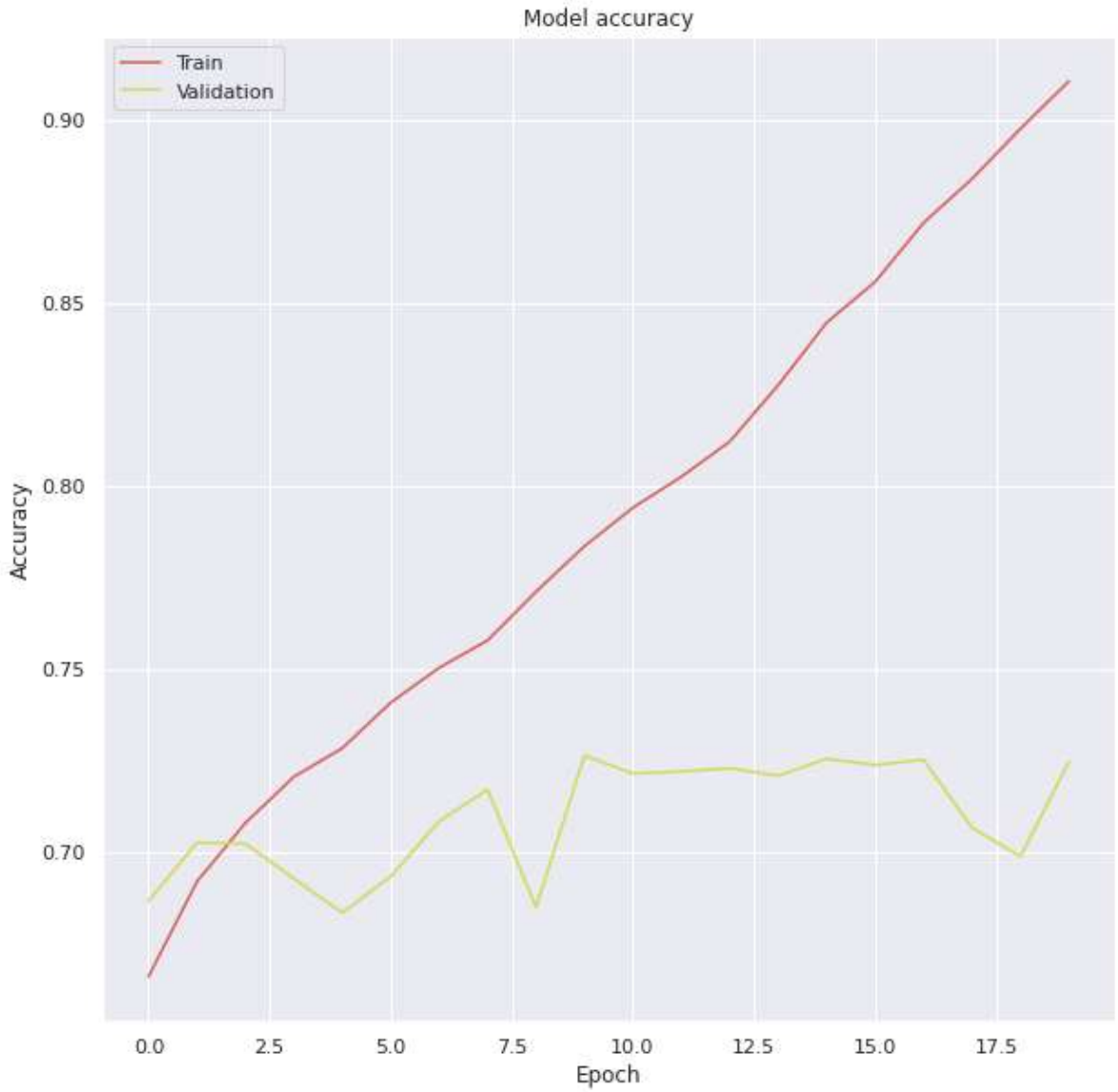
```
history_2 = model.fit(x = X_augmented, y = y_augmented, epochs = 20, validation_split = 0.1
```

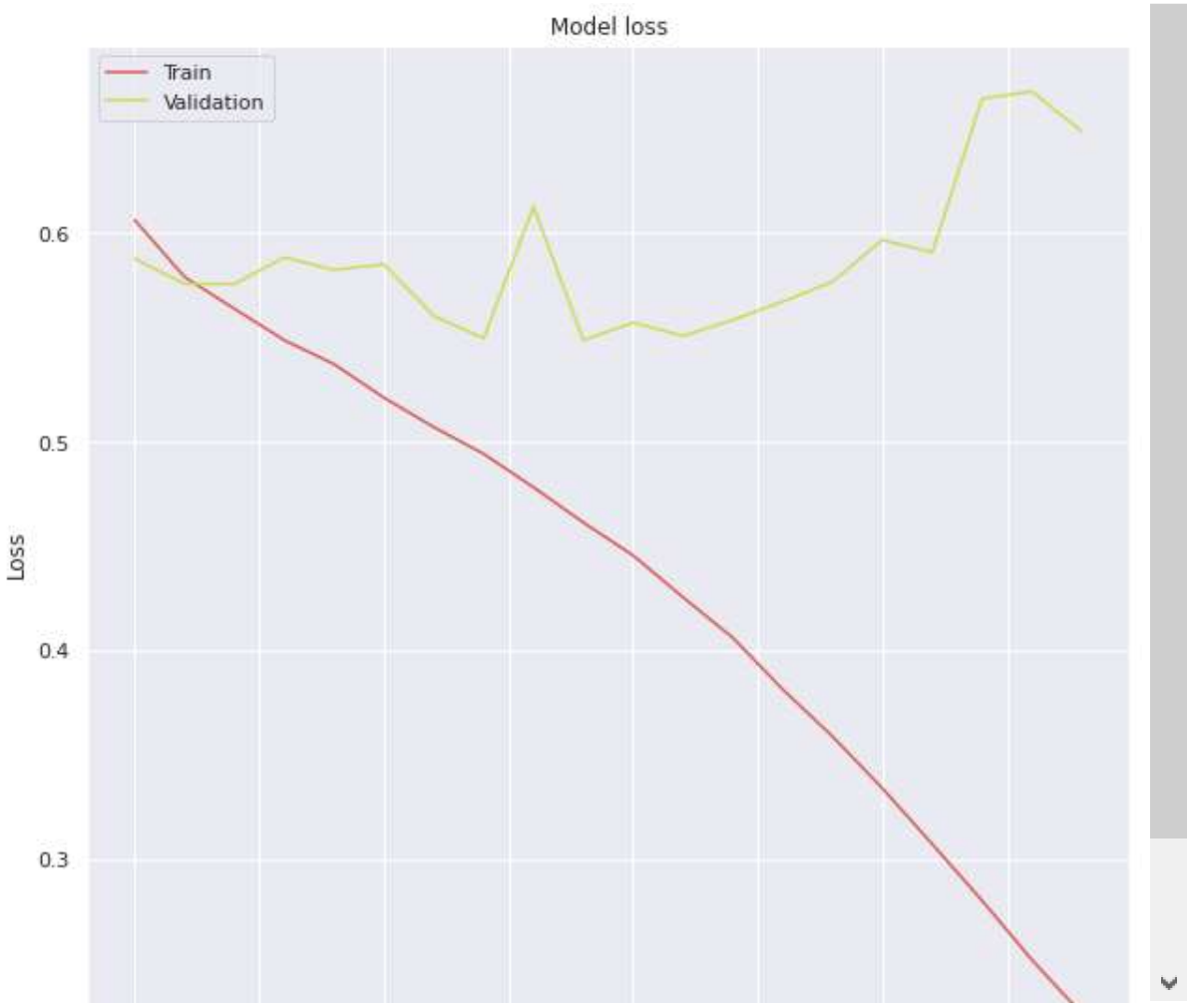
```
Epoch 1/20
611/611 [==============================] - 5s 8ms/step - loss: 0.6061 - accu
racy: 0.6660 - val_loss: 0.5876 - val_accuracy: 0.6867
Epoch 2/20
611/611 [==============================] - 5s 7ms/step - loss: 0.5788 - accu
racy: 0.6920 - val_loss: 0.5756 - val_accuracy: 0.7026
Epoch 3/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5636 - accu
racy: 0.7080 - val_loss: 0.5755 - val_accuracy: 0.7023
Epoch 4/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5486 - accu
racy: 0.7206 - val_loss: 0.5882 - val_accuracy: 0.6928
Epoch 5/20
611/611 [==============================] - 5s 7ms/step - loss: 0.5371 - accu
racy: 0.7284 - val_loss: 0.5822 - val_accuracy: 0.6835
Epoch 6/20
611/611 [==============================] - 4s 7ms/step - loss: 0.5210 - accu
racy: 0.7408 - val_loss: 0.5849 - val_accuracy: 0.6933
Epoch 7/20
611/611 [==============================] - 5s 7ms/step - loss: 0.5070 - accu
racy: 0.7503 - val_loss: 0.5601 - val_accuracy: 0.7084
Epoch 8/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4942 - accu
racy: 0.7578 - val_loss: 0.5495 - val_accuracy: 0.7171
Epoch 9/20
611/611 [==============================] - 5s 7ms/step - loss: 0.4781 - accu
racy: 0.7711 - val_loss: 0.6124 - val_accuracy: 0.6849
Epoch 10/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4613 - accu
racy: 0.7835 - val_loss: 0.5486 - val_accuracy: 0.7264
Epoch 11/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4455 - accu
racy: 0.7940 - val_loss: 0.5570 - val_accuracy: 0.7214
Epoch 12/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4255 - accu
racy: 0.8025 - val_loss: 0.5506 - val_accuracy: 0.7220
Epoch 13/20
611/611 [==============================] - 4s 7ms/step - loss: 0.4060 - accu
racy: 0.8121 - val_loss: 0.5583 - val_accuracy: 0.7229
Epoch 14/20
611/611 [==============================] - 4s 7ms/step - loss: 0.3814 - accu
racy: 0.8275 - val_loss: 0.5670 - val_accuracy: 0.7209
Epoch 15/20
611/611 [==============================] - 4s 7ms/step - loss: 0.3588 - accu
racy: 0.8446 - val_loss: 0.5766 - val_accuracy: 0.7255
Epoch 16/20
611/611 [==============================] - 4s 7ms/step - loss: 0.3340 - accu
racy: 0.8557 - val_loss: 0.5965 - val_accuracy: 0.7238
Epoch 17/20
611/611 [==============================] - 5s 7ms/step - loss: 0.3073 - accu
racy: 0.8718 - val_loss: 0.5907 - val_accuracy: 0.7252
Epoch 18/20
611/611 [==============================] - 4s 7ms/step - loss: 0.2803 - accu
racy: 0.8838 - val_loss: 0.6643 - val_accuracy: 0.7067
Epoch 19/20
611/611 [==============================] - 4s 7ms/step - loss: 0.2519 - accu
racy: 0.8974 - val_loss: 0.6676 - val_accuracy: 0.6988
```

```
Epoch 20/20
611/611 [==============================] - 4s 7ms/step - loss: 0.2260 - accu
racy: 0.9104 - val_loss: 0.6485 - val_accuracy: 0.7246
```

In [27]:

```
plt.plot(history_2.history['accuracy'])
plt.plot(history_2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```





In [28]:

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

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

```
63/63 [==============================] - 0s 4ms/step - loss: 0.6343 - accura
cy: 0.7535
Test loss, test accuracy: [0.6342598795890808, 0.7534999847412109]
```

После того, как сеть обучилась на тех же данных, к которым был применён data augmentation, точность предсказания даже немного уменьшилась — 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 [1]:

```
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 [7]:

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

Out[7]:

```
((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 [12]:

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

! pip show tensorflow-gpu
```

```
     |████████████████████████████████| 516.2MB 30kB/s
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: gast, numpy, astunparse, tensorboard, tensorflow-estimator, termco
lor, wrapt, google-pasta, opt-einsum, grpcio, scipy, h5py, six, keras-prepro
cessing, absl-py, protobuf, wheel
Required-by:
```

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('ir
y_test = to_categorical(test_df_reshaped['label'].astype('category').cat.codes.astype('int3
```

In [15]:

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

Out[15]:

```
((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 =
                 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 =
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 [19]:

```
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 [20]:

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

```
Epoch 1/20
730/730 [==============================] - 3s 4ms/step - loss: 1.2839 - cate
gorical_accuracy: 0.6213 - val_loss: 0.6058 - val_categorical_accuracy: 0.82
37
Epoch 2/20
730/730 [==============================] - 3s 4ms/step - loss: 0.3387 - cate
gorical_accuracy: 0.9224 - val_loss: 0.1501 - val_categorical_accuracy: 0.98
52
Epoch 3/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0872 - cate
gorical_accuracy: 0.9943 - val_loss: 0.0430 - val_categorical_accuracy: 0.99
98
Epoch 4/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0279 - cate
gorical_accuracy: 0.9999 - val_loss: 0.0172 - val_categorical_accuracy: 1.00
00
Epoch 5/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0119 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0081 - val_categorical_accuracy: 1.00
00
Epoch 6/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0065 - cate
gorical_accuracy: 0.9999 - val_loss: 0.0046 - val_categorical_accuracy: 1.00
00
Epoch 7/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0035 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0026 - val_categorical_accuracy: 1.00
00
Epoch 8/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0021 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0016 - val_categorical_accuracy: 1.00
00
Epoch 9/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0013 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0011 - val_categorical_accuracy: 1.00
00
Epoch 10/20
730/730 [==============================] - 3s 4ms/step - loss: 8.4052e-04 -
categorical_accuracy: 1.0000 - val_loss: 6.6852e-04 - val_categorical_accura
cy: 1.0000
Epoch 11/20
730/730 [==============================] - 3s 4ms/step - loss: 5.4248e-04 -
categorical_accuracy: 1.0000 - val_loss: 4.6793e-04 - val_categorical_accura
cy: 1.0000
Epoch 12/20
730/730 [==============================] - 3s 4ms/step - loss: 3.6096e-04 -
categorical_accuracy: 1.0000 - val_loss: 2.9138e-04 - val_categorical_accura
cy: 1.0000
Epoch 13/20
730/730 [==============================] - 3s 4ms/step - loss: 2.3311e-04 -
categorical_accuracy: 1.0000 - val_loss: 1.9329e-04 - val_categorical_accura
cy: 1.0000
Epoch 14/20
730/730 [==============================] - 3s 4ms/step - loss: 1.5398e-04 -
categorical_accuracy: 1.0000 - val_loss: 1.3504e-04 - val_categorical_accura
cy: 1.0000
Epoch 15/20
```

```
730/730 [==============================] - 3s 4ms/step - loss: 1.0419e-04 -
categorical_accuracy: 1.0000 - val_loss: 8.6794e-05 - val_categorical_accura
cy: 1.0000
Epoch 16/20
730/730 [==============================] - 3s 4ms/step - loss: 7.0365e-05 -
categorical_accuracy: 1.0000 - val_loss: 5.9591e-05 - val_categorical_accura
cy: 1.0000
Epoch 17/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0283 - cate
gorical_accuracy: 0.9922 - val_loss: 9.3350e-04 - val_categorical_accuracy:
1.0000
Epoch 18/20
730/730 [==============================] - 3s 4ms/step - loss: 5.7962e-04 -
categorical_accuracy: 1.0000 - val_loss: 4.6569e-04 - val_categorical_accura
cy: 1.0000
Epoch 19/20
730/730 [==============================] - 3s 4ms/step - loss: 3.4192e-04 -
categorical_accuracy: 1.0000 - val_loss: 3.1113e-04 - val_categorical_accura
cy: 1.0000
Epoch 20/20
730/730 [==============================] - 3s 4ms/step - loss: 2.3316e-04 -
categorical_accuracy: 1.0000 - val_loss: 2.1958e-04 - val_categorical_accura
cy: 1.0000
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

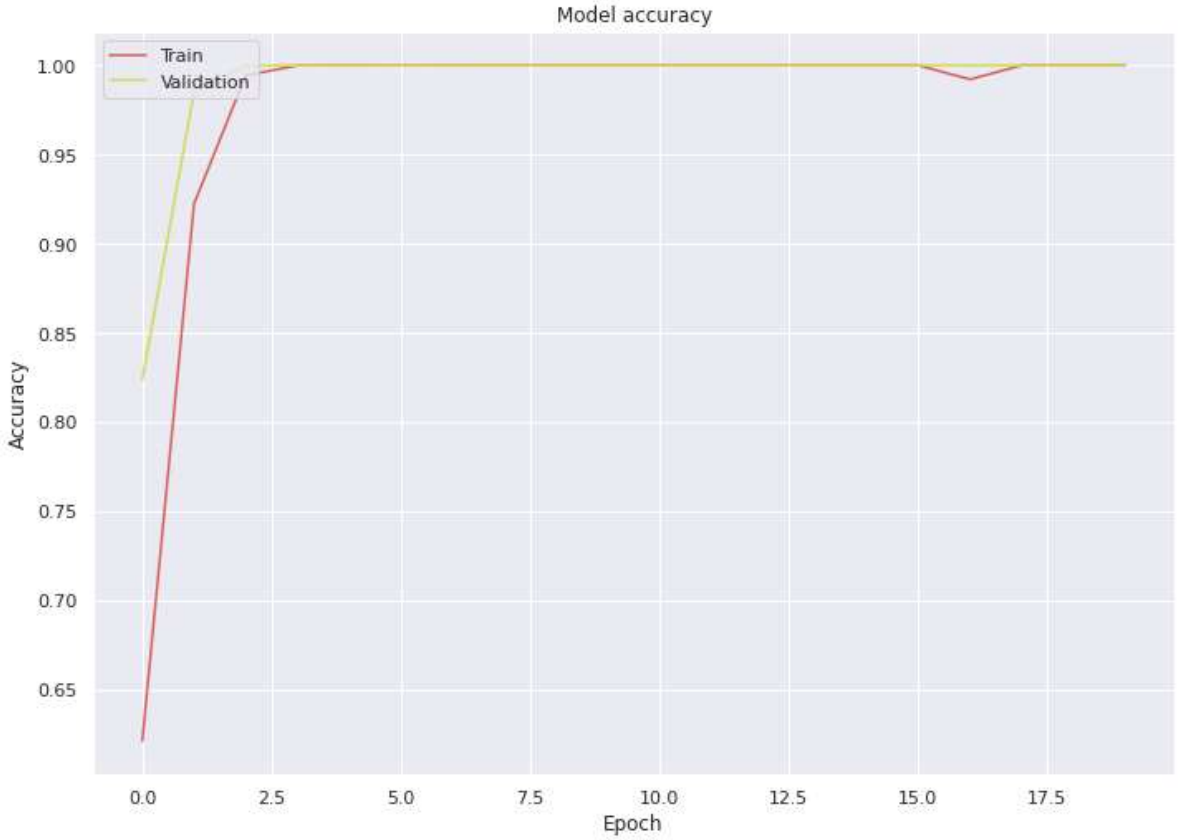
In [22]:

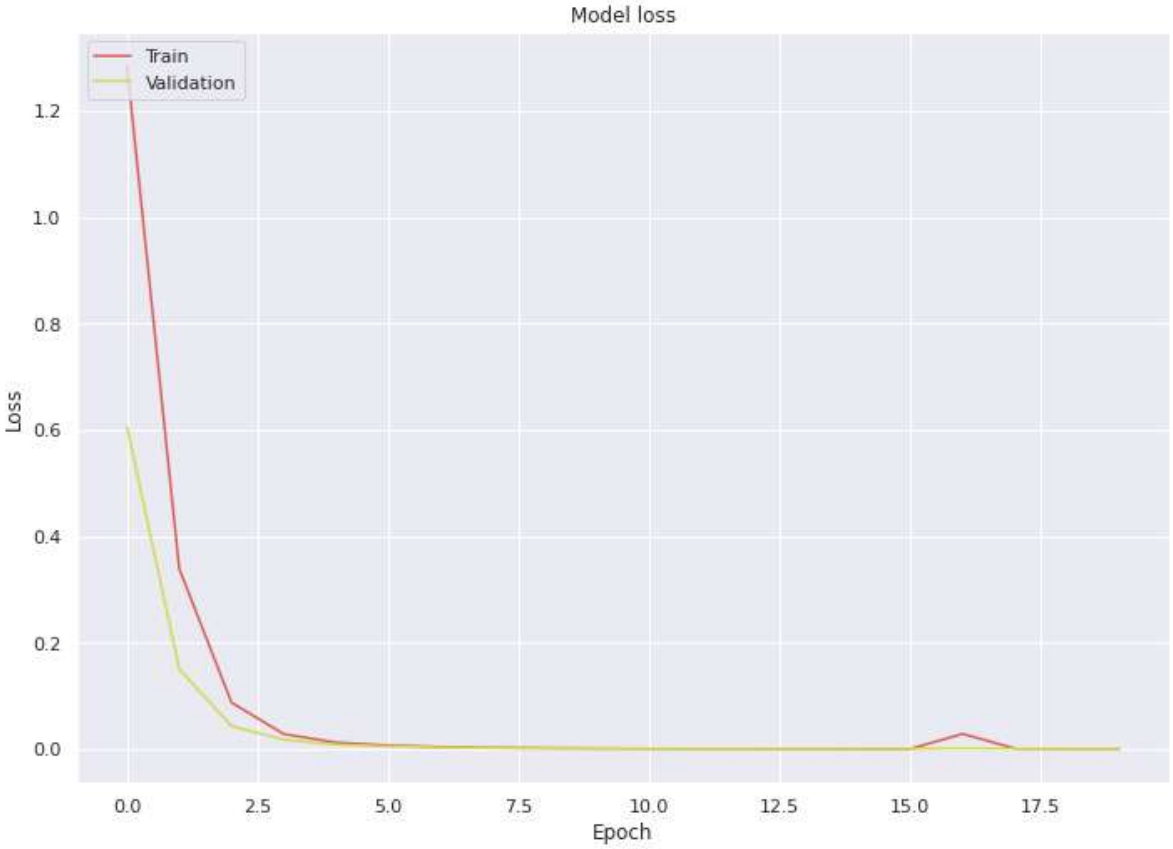
```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

In [23]:

```
plt.plot(history.history['categorical_accuracy'])
plt.plot(history.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

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 = 'upper left')
plt.show()
```





In [24]:

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

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

```
225/225 [==============================] - 1s 2ms/step - loss: 0.7200 - cate
gorical_accuracy: 0.8441
Test loss, test accuracy: [0.720040500164032, 0.8441160321235657]
```

За 20 эпох удалось достичь точности 84% 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, 1])

  return image.numpy()
```

In [26]:

```
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[26]:

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

In [0]:

```
y_train_augmented = y_train
```

In [28]:

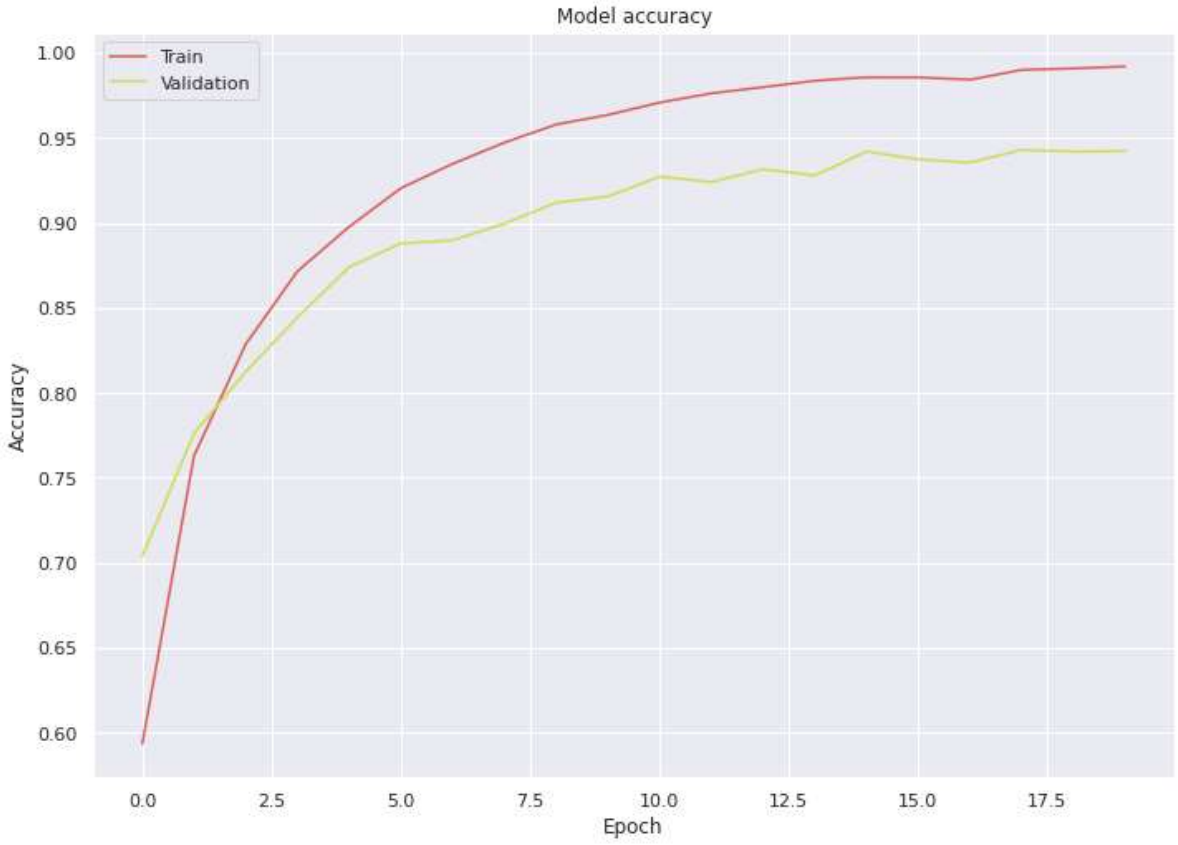
```
history_2 = model.fit(x = X_train_augmented, y = y_train_augmented, epochs = 20, validatior
```

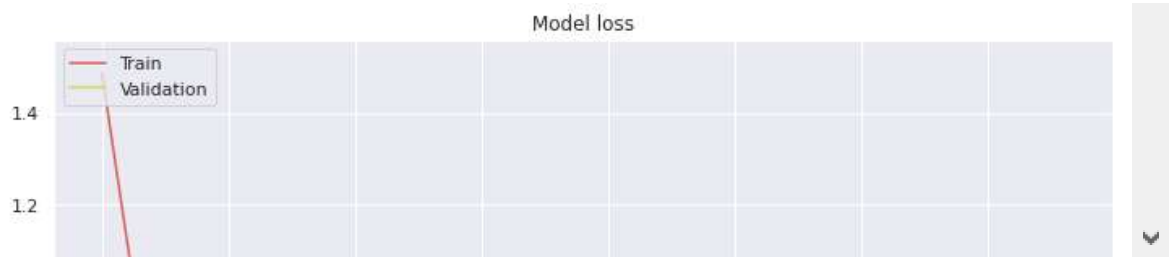
```
Epoch 1/20
730/730 [==============================] - 3s 4ms/step - loss: 1.4882 - cate
gorical_accuracy: 0.5935 - val_loss: 0.9350 - val_categorical_accuracy: 0.70
38
Epoch 2/20
730/730 [==============================] - 3s 4ms/step - loss: 0.7458 - cate
gorical_accuracy: 0.7629 - val_loss: 0.7156 - val_categorical_accuracy: 0.77
62
Epoch 3/20
730/730 [==============================] - 3s 4ms/step - loss: 0.5453 - cate
gorical_accuracy: 0.8287 - val_loss: 0.5752 - val_categorical_accuracy: 0.81
23
Epoch 4/20
730/730 [==============================] - 3s 4ms/step - loss: 0.4258 - cate
gorical_accuracy: 0.8715 - val_loss: 0.4928 - val_categorical_accuracy: 0.84
44
Epoch 5/20
730/730 [==============================] - 3s 4ms/step - loss: 0.3402 - cate
gorical_accuracy: 0.8977 - val_loss: 0.4242 - val_categorical_accuracy: 0.87
40
Epoch 6/20
730/730 [==============================] - 3s 4ms/step - loss: 0.2777 - cate
gorical_accuracy: 0.9204 - val_loss: 0.3704 - val_categorical_accuracy: 0.88
78
Epoch 7/20
730/730 [==============================] - 3s 4ms/step - loss: 0.2303 - cate
gorical_accuracy: 0.9347 - val_loss: 0.3587 - val_categorical_accuracy: 0.88
98
Epoch 8/20
730/730 [==============================] - 3s 4ms/step - loss: 0.1896 - cate
gorical_accuracy: 0.9472 - val_loss: 0.3076 - val_categorical_accuracy: 0.89
95
Epoch 9/20
730/730 [==============================] - 3s 4ms/step - loss: 0.1572 - cate
gorical_accuracy: 0.9579 - val_loss: 0.2898 - val_categorical_accuracy: 0.91
19
Epoch 10/20
730/730 [==============================] - 3s 4ms/step - loss: 0.1344 - cate
gorical_accuracy: 0.9635 - val_loss: 0.2764 - val_categorical_accuracy: 0.91
55
Epoch 11/20
730/730 [==============================] - 3s 4ms/step - loss: 0.1129 - cate
gorical_accuracy: 0.9708 - val_loss: 0.2405 - val_categorical_accuracy: 0.92
72
Epoch 12/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0956 - cate
gorical_accuracy: 0.9762 - val_loss: 0.2428 - val_categorical_accuracy: 0.92
40
Epoch 13/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0815 - cate
gorical_accuracy: 0.9798 - val_loss: 0.2197 - val_categorical_accuracy: 0.93
15
Epoch 14/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0701 - cate
gorical_accuracy: 0.9836 - val_loss: 0.2292 - val_categorical_accuracy: 0.92
79
Epoch 15/20
```

```
730/730 [==============================] - 3s 4ms/step - loss: 0.0619 - cate
gorical_accuracy: 0.9856 - val_loss: 0.2041 - val_categorical_accuracy: 0.94
20
Epoch 16/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0572 - cate
gorical_accuracy: 0.9856 - val_loss: 0.2073 - val_categorical_accuracy: 0.93
74
Epoch 17/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0584 - cate
gorical_accuracy: 0.9842 - val_loss: 0.2021 - val_categorical_accuracy: 0.93
54
Epoch 18/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0446 - cate
gorical_accuracy: 0.9900 - val_loss: 0.1930 - val_categorical_accuracy: 0.94
29
Epoch 19/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0402 - cate
gorical_accuracy: 0.9909 - val_loss: 0.2038 - val_categorical_accuracy: 0.94
20
Epoch 20/20
730/730 [==============================] - 3s 4ms/step - loss: 0.0364 - cate
gorical_accuracy: 0.9920 - val_loss: 0.2035 - val_categorical_accuracy: 0.94
22
```

In [29]:

```
plt.plot(history_2.history['categorical_accuracy'])
plt.plot(history_2.history['val_categorical_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```





In [30]:

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

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

```
225/225 [==============================] - 1s 3ms/step - loss: 0.3715 - cate
gorical_accuracy: 0.9106
Test loss, test accuracy: [0.371463805437088, 0.910624623298645]
```

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 [1]:

```
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 [8]:

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

Out[8]:

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

In [9]:

```
import nltk

nltk.download('punkt')
```

```
[nltk_data] Downloading package punkt to /root/nltk_data...
[nltk_data]   Unzipping tokenizers/punkt.zip.
```

Out[9]:

```
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('[' + stri

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):

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

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

    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 [18]:

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

! pip show tensorflow-gpu
```

```
     |████████████████████████████████| 516.2MB 30kB/s
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: wheel, google-pasta, grpcio, opt-einsum, keras-preprocessing, scip
y, tensorflow-estimator, six, astunparse, numpy, termcolor, wrapt, tensorboa
rd, gast, h5py, absl-py, protobuf
Required-by:
```

In [0]:

```
import tensorflow as tf
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 [21]:

```
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'))
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:From /usr/local/lib/python3.6/dist-packages/tensorflow/py
thon/ops/resource_variable_ops.py:1666: calling BaseResourceVariable.__init_
_ (from tensorflow.python.ops.resource_variable_ops) with constraint is depr
ecated and will be removed in a future version.
Instructions for updating:
If using Keras pass *_constraint arguments to layers.
```

In [22]:

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

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

In [24]:

```
history = model.fit(x = X_train_intized, y = y_train_intized, validation_split = 0.15, epoc
```

```
Train on 38250 samples, validate on 6750 samples
Epoch 1/20
38250/38250 [==============================] - 50s 1ms/sample - loss: 0.6937
- accuracy: 0.5199 - val_loss: 0.6879 - val_accuracy: 0.5447
Epoch 2/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6883
- accuracy: 0.5420 - val_loss: 0.6892 - val_accuracy: 0.5348
Epoch 3/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6859
- accuracy: 0.5495 - val_loss: 0.6883 - val_accuracy: 0.5519
Epoch 4/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6842
- accuracy: 0.5548 - val_loss: 0.6832 - val_accuracy: 0.5538
Epoch 5/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6814
- accuracy: 0.5604 - val_loss: 0.6834 - val_accuracy: 0.5567
Epoch 6/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6795
- accuracy: 0.5670 - val_loss: 0.6835 - val_accuracy: 0.5545
Epoch 7/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6769
- accuracy: 0.5690 - val_loss: 0.6813 - val_accuracy: 0.5609
Epoch 8/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6750
- accuracy: 0.5748 - val_loss: 0.6784 - val_accuracy: 0.5671
Epoch 9/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6728
- accuracy: 0.5766 - val_loss: 0.6777 - val_accuracy: 0.5665
Epoch 10/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6705
- accuracy: 0.5816 - val_loss: 0.6831 - val_accuracy: 0.5585
Epoch 11/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6692
- accuracy: 0.5856 - val_loss: 0.6794 - val_accuracy: 0.5667
Epoch 12/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6664
- accuracy: 0.5882 - val_loss: 0.6751 - val_accuracy: 0.5721
Epoch 13/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6653
- accuracy: 0.5893 - val_loss: 0.6769 - val_accuracy: 0.5698
Epoch 14/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6626
- accuracy: 0.5925 - val_loss: 0.6735 - val_accuracy: 0.5757
Epoch 15/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6606
- accuracy: 0.5983 - val_loss: 0.6833 - val_accuracy: 0.5676
Epoch 16/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6592
- accuracy: 0.5974 - val_loss: 0.6769 - val_accuracy: 0.5702
Epoch 17/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6559
- accuracy: 0.6018 - val_loss: 0.6763 - val_accuracy: 0.5751
Epoch 18/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6526
- accuracy: 0.6098 - val_loss: 0.6780 - val_accuracy: 0.5763
Epoch 19/20
38250/38250 [==============================] - 48s 1ms/sample - loss: 0.6498
```

```
- accuracy: 0.6117 - val_loss: 0.6763 - val_accuracy: 0.5809
Epoch 20/20
38250/38250 [==============================] - 49s 1ms/sample - loss: 0.6467
- accuracy: 0.6119 - val_loss: 0.6789 - val_accuracy: 0.5727
```

In [0]:

```
%matplotlib inline

import matplotlib.pyplot as plt
```

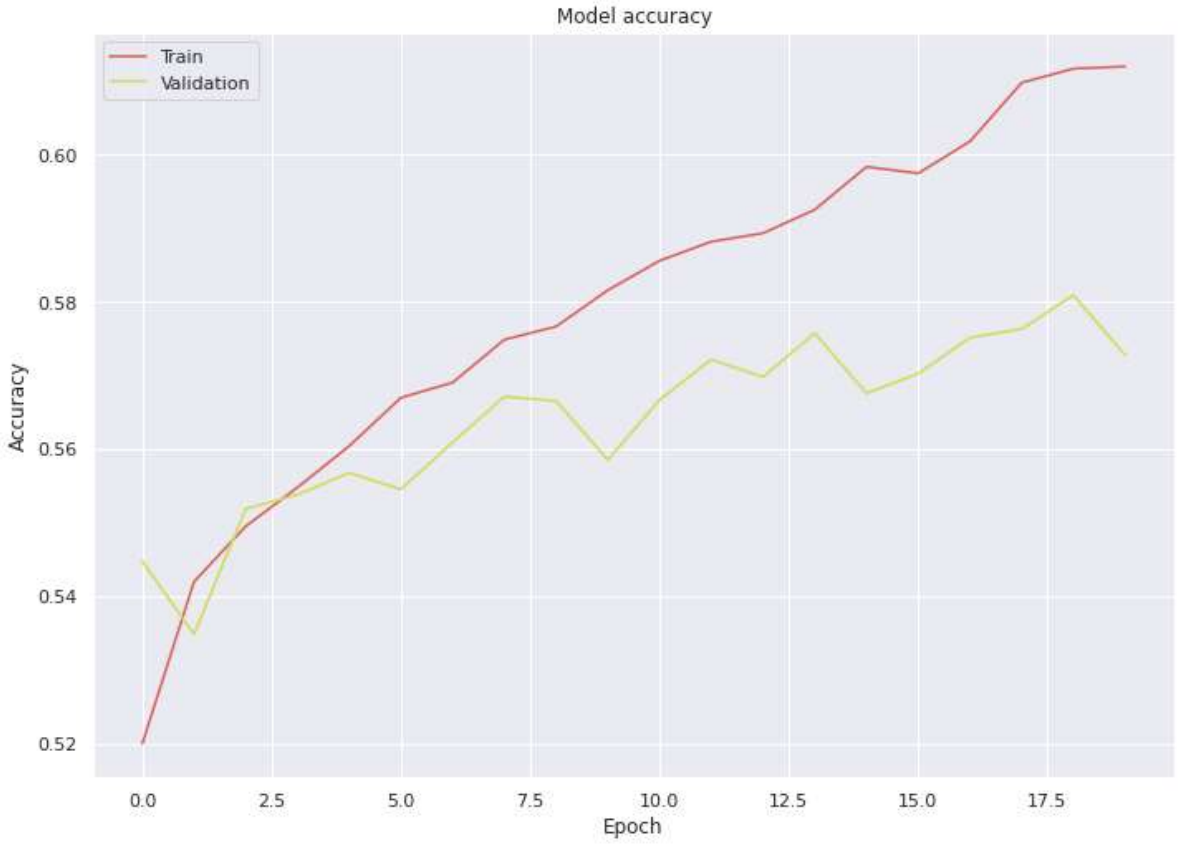
In [26]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

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

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

In [27]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

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 = 'upper left')
plt.show()
```



In [28]:

```
results = model.evaluate(X_test_intized, y_test_intized)

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

```
Test loss, test accuracy: [0.682829614162445, 0.5698]
```

На валидационной выборке удалось достичь точности 56%.

Задание 3

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

Используем 300-мерные вектора *FastTest* — 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] == _

#     return actual_vectors_
```

In [0]:

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

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 [40]:

```
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'))
```

```
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
```

In [41]:

```
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 [42]:

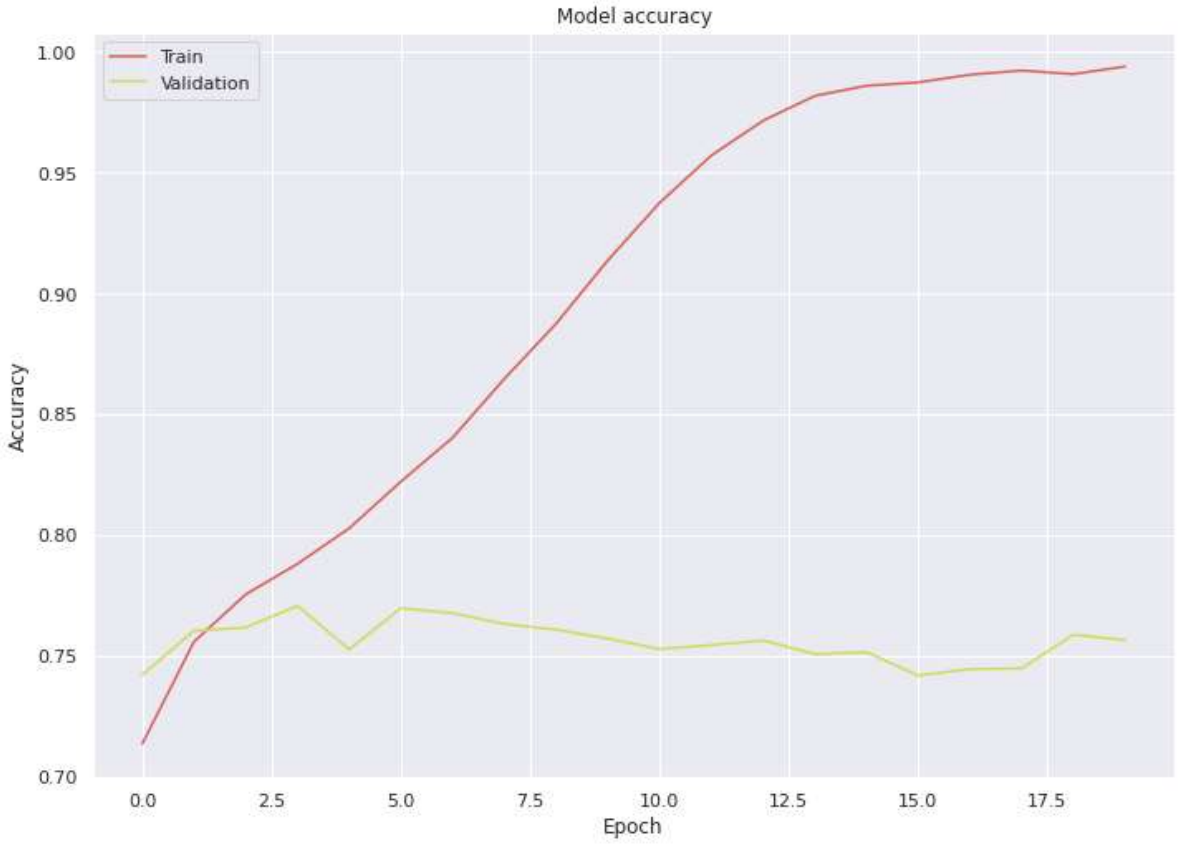
```
history_2 = model_2.fit(x = X_train_vectorized, y = y_train_vectorized, validation_split =
```

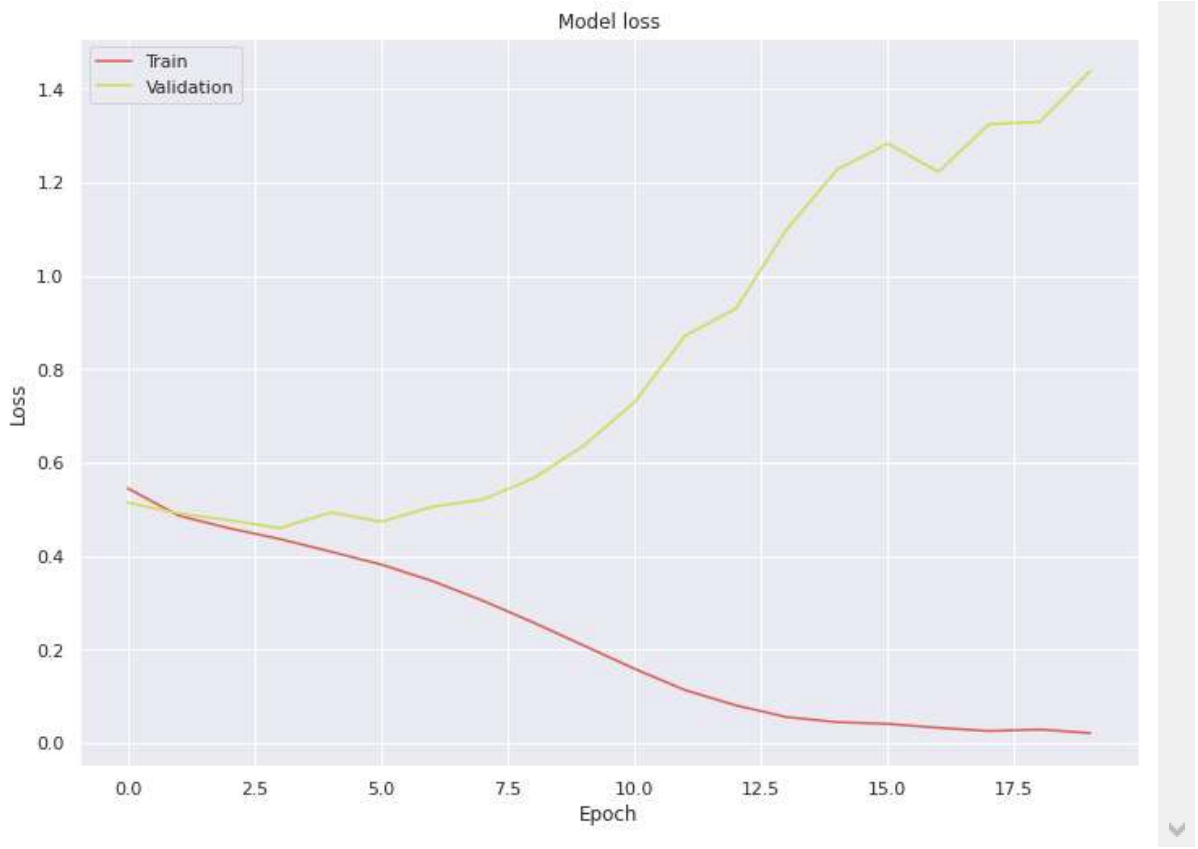
```
Train on 38250 samples, validate on 6750 samples
Epoch 1/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.5449
- accuracy: 0.7137 - val_loss: 0.5143 - val_accuracy: 0.7422
Epoch 2/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.4866
- accuracy: 0.7559 - val_loss: 0.4913 - val_accuracy: 0.7604
Epoch 3/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.4594
- accuracy: 0.7755 - val_loss: 0.4766 - val_accuracy: 0.7618
Epoch 4/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.4364
- accuracy: 0.7881 - val_loss: 0.4598 - val_accuracy: 0.7708
Epoch 5/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.4095
- accuracy: 0.8028 - val_loss: 0.4932 - val_accuracy: 0.7527
Epoch 6/20
38250/38250 [==============================] - 53s 1ms/sample - loss: 0.3818
- accuracy: 0.8222 - val_loss: 0.4736 - val_accuracy: 0.7698
Epoch 7/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.3469
- accuracy: 0.8403 - val_loss: 0.5053 - val_accuracy: 0.7677
Epoch 8/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.3047
- accuracy: 0.8647 - val_loss: 0.5212 - val_accuracy: 0.7633
Epoch 9/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.2579
- accuracy: 0.8875 - val_loss: 0.5661 - val_accuracy: 0.7609
Epoch 10/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.2083
- accuracy: 0.9137 - val_loss: 0.6364 - val_accuracy: 0.7572
Epoch 11/20
38250/38250 [==============================] - 52s 1ms/sample - loss: 0.1586
- accuracy: 0.9374 - val_loss: 0.7294 - val_accuracy: 0.7529
Epoch 12/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.1128
- accuracy: 0.9570 - val_loss: 0.8719 - val_accuracy: 0.7545
Epoch 13/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0804
- accuracy: 0.9714 - val_loss: 0.9297 - val_accuracy: 0.7564
Epoch 14/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0552
- accuracy: 0.9817 - val_loss: 1.0989 - val_accuracy: 0.7508
Epoch 15/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0443
- accuracy: 0.9859 - val_loss: 1.2276 - val_accuracy: 0.7516
Epoch 16/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0406
- accuracy: 0.9872 - val_loss: 1.2833 - val_accuracy: 0.7419
Epoch 17/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0322
- accuracy: 0.9904 - val_loss: 1.2227 - val_accuracy: 0.7446
Epoch 18/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0253
- accuracy: 0.9922 - val_loss: 1.3247 - val_accuracy: 0.7449
Epoch 19/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0286
```

```
- accuracy: 0.9907 - val_loss: 1.3299 - val_accuracy: 0.7588
Epoch 20/20
38250/38250 [==============================] - 51s 1ms/sample - loss: 0.0209
- accuracy: 0.9938 - val_loss: 1.4388 - val_accuracy: 0.7566
```

In [43]:

```
plt.plot(history_2.history['accuracy'])
plt.plot(history_2.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

plt.plot(history_2.history['loss'])
plt.plot(history_2.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```





In [44]:

```
results_2 = model_2.evaluate(X_test_vectorized, y_test_vectorized)

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

```
Test loss, test accuracy: [1.4954523480892181, 0.7468]
```

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 [45]:

```
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'))
```

```
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_2 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
WARNING:tensorflow:Layer lstm_3 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
```

In [46]:

```
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 [50]:

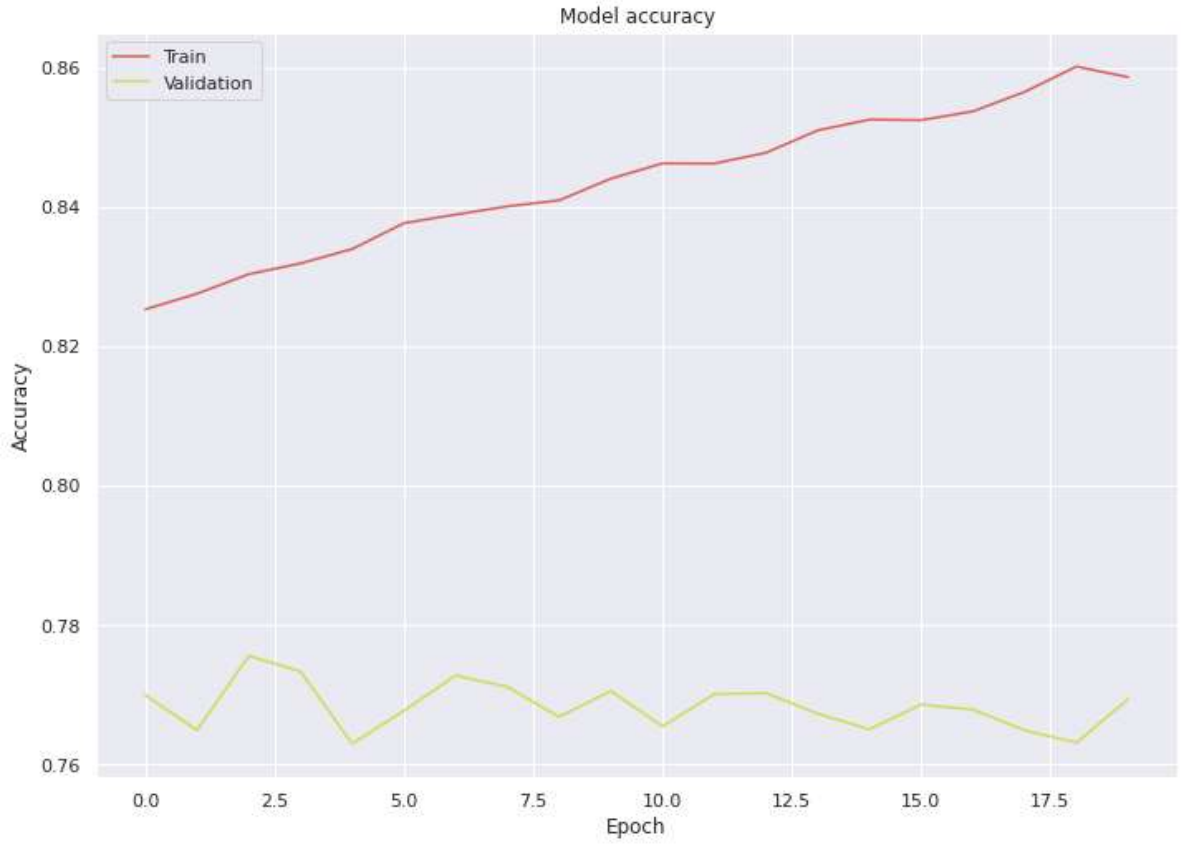
```
history_3 = model_3.fit(x = X_train_vectorized, y = y_train_vectorized, validation_split =
```

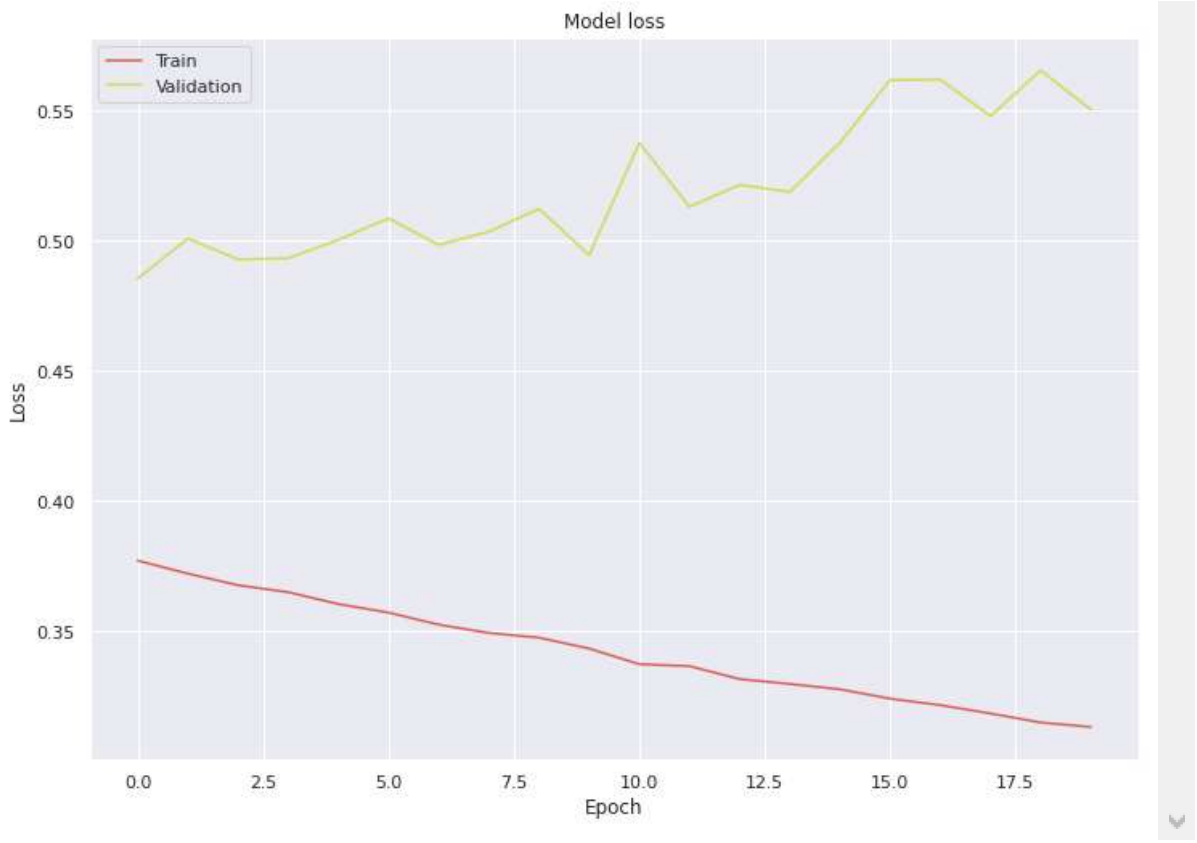
```
Train on 38250 samples, validate on 6750 samples
Epoch 1/20
38250/38250 [==============================] - 84s 2ms/sample - loss: 0.3770
- accuracy: 0.8253 - val_loss: 0.4855 - val_accuracy: 0.7699
Epoch 2/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.3720
- accuracy: 0.8275 - val_loss: 0.5009 - val_accuracy: 0.7649
Epoch 3/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.3676
- accuracy: 0.8303 - val_loss: 0.4929 - val_accuracy: 0.7756
Epoch 4/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.3649
- accuracy: 0.8318 - val_loss: 0.4934 - val_accuracy: 0.7733
Epoch 5/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.3603
- accuracy: 0.8339 - val_loss: 0.5004 - val_accuracy: 0.7630
Epoch 6/20
38250/38250 [==============================] - 81s 2ms/sample - loss: 0.3571
- accuracy: 0.8376 - val_loss: 0.5087 - val_accuracy: 0.7677
Epoch 7/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.3524
- accuracy: 0.8388 - val_loss: 0.4985 - val_accuracy: 0.7727
Epoch 8/20
38250/38250 [==============================] - 85s 2ms/sample - loss: 0.3492
- accuracy: 0.8400 - val_loss: 0.5036 - val_accuracy: 0.7711
Epoch 9/20
38250/38250 [==============================] - 85s 2ms/sample - loss: 0.3474
- accuracy: 0.8409 - val_loss: 0.5124 - val_accuracy: 0.7668
Epoch 10/20
38250/38250 [==============================] - 84s 2ms/sample - loss: 0.3432
- accuracy: 0.8440 - val_loss: 0.4945 - val_accuracy: 0.7705
Epoch 11/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.3372
- accuracy: 0.8462 - val_loss: 0.5378 - val_accuracy: 0.7655
Epoch 12/20
38250/38250 [==============================] - 84s 2ms/sample - loss: 0.3365
- accuracy: 0.8462 - val_loss: 0.5131 - val_accuracy: 0.7701
Epoch 13/20
38250/38250 [==============================] - 84s 2ms/sample - loss: 0.3315
- accuracy: 0.8477 - val_loss: 0.5216 - val_accuracy: 0.7702
Epoch 14/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.3296
- accuracy: 0.8509 - val_loss: 0.5189 - val_accuracy: 0.7673
Epoch 15/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.3275
- accuracy: 0.8525 - val_loss: 0.5378 - val_accuracy: 0.7650
Epoch 16/20
38250/38250 [==============================] - 84s 2ms/sample - loss: 0.3240
- accuracy: 0.8524 - val_loss: 0.5619 - val_accuracy: 0.7686
Epoch 17/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.3214
- accuracy: 0.8536 - val_loss: 0.5621 - val_accuracy: 0.7679
Epoch 18/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.3182
- accuracy: 0.8564 - val_loss: 0.5480 - val_accuracy: 0.7649
Epoch 19/20
38250/38250 [==============================] - 83s 2ms/sample - loss: 0.3148
```

```
- accuracy: 0.8601 - val_loss: 0.5656 - val_accuracy: 0.7631
Epoch 20/20
38250/38250 [==============================] - 82s 2ms/sample - loss: 0.3131
- accuracy: 0.8586 - val_loss: 0.5506 - val_accuracy: 0.7693
```

In [51]:

```
plt.plot(history_3.history['accuracy'])
plt.plot(history_3.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

plt.plot(history_3.history['loss'])
plt.plot(history_3.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()
```





In [52]:

```
results_3 = model_3.evaluate(X_test_vectorized, y_test_vectorized)

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

```
Test loss, test accuracy: [0.5695644823074341, 0.7558]
```

Добавление ещё одного рекуррентного слоя ненамного улучшило результат — точность 75% 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 достигнуть максимальной точности 76%.

Лабораторная работа №8

Рекуррентные нейронные сети для анализа временных рядов

Набор данных для прогнозирования временных рядов, который состоит из среднемесячного числа пятен на солнце, наблюдаемых с января 1749 по август 2017.

Данные в виде csv-файла можно скачать на сайте *Kaggle*: https://www.kaggle.com/robervalt/sunspots/ (https://www.kaggle.com/robervalt/sunspots/)

Задание 1

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

In [1]:

```
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 [7]:

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

```
(3252, 2)
```

In [8]:

```
all_df.keys()
```

Out[8]:

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

In [9]:

```
from statsmodels.tsa.seasonal import seasonal_decompose

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

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

In [0]:

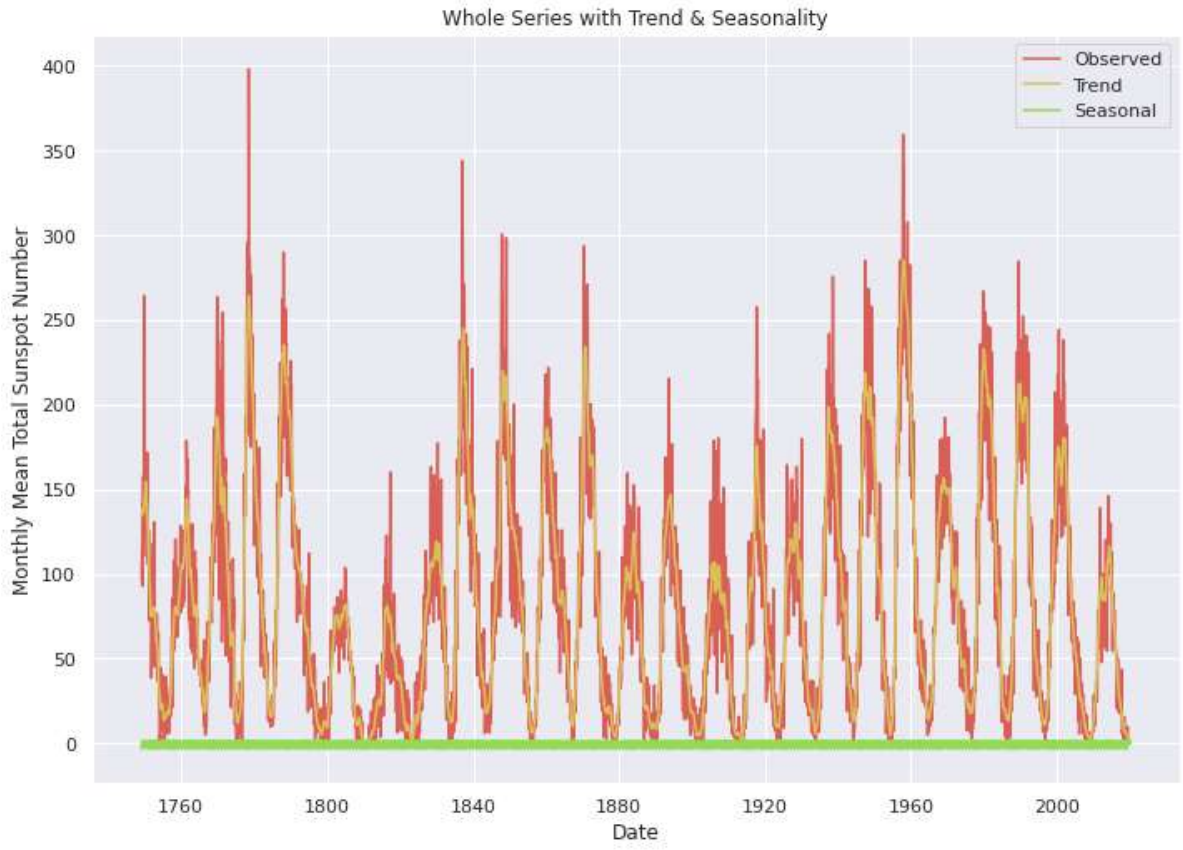
```
%matplotlib inline

import matplotlib.pyplot as plt
```

In [0]:

```
import seaborn as sns

from matplotlib import rcParams

rcParams['figure.figsize'] = 11.7, 8.27

sns.set()

sns.set_palette(sns.color_palette('hls', 8))
```

In [12]:

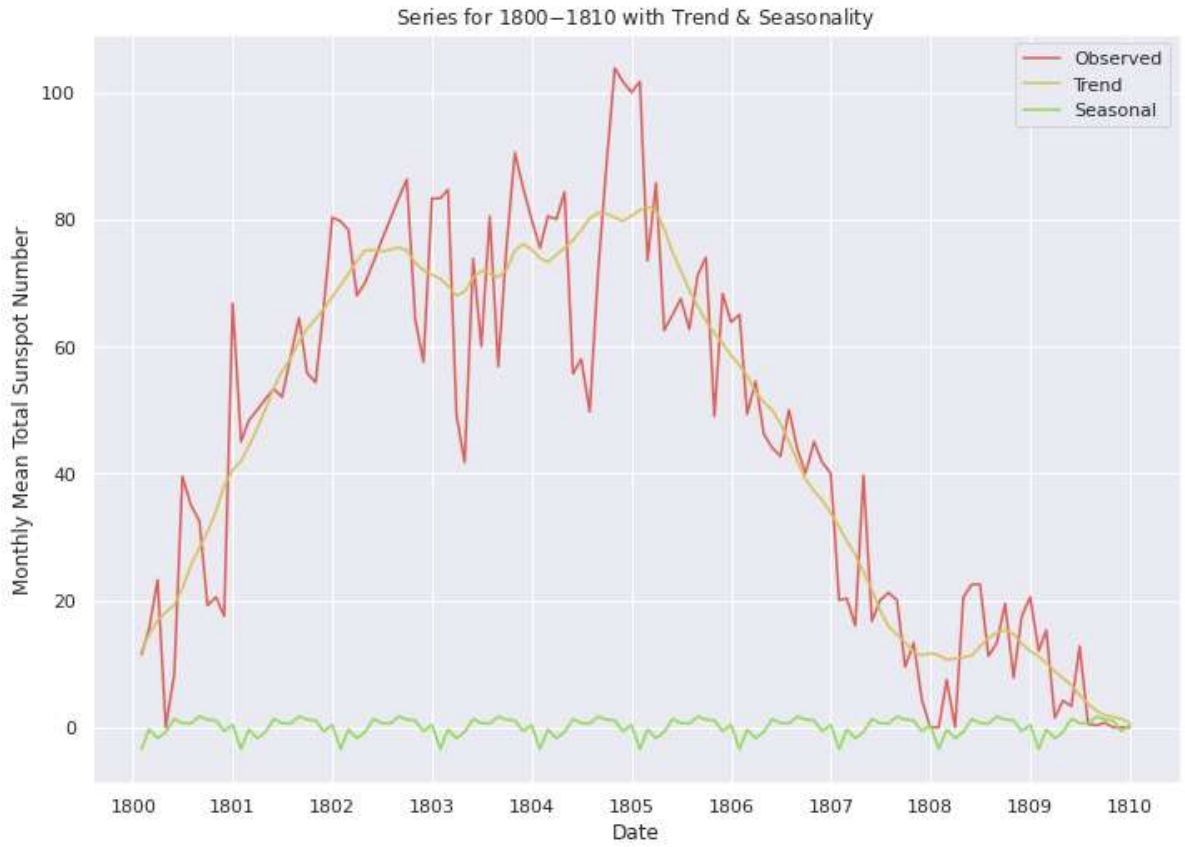
```
sns.lineplot(data = additive.observed, label = 'Observed')
sns.lineplot(data = additive.trend, label = 'Trend')
sns.lineplot(data = additive.seasonal, label = 'Seasonal')

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

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

plt.show()
```



Рассмотрим подробнее на небольшом промежутке:

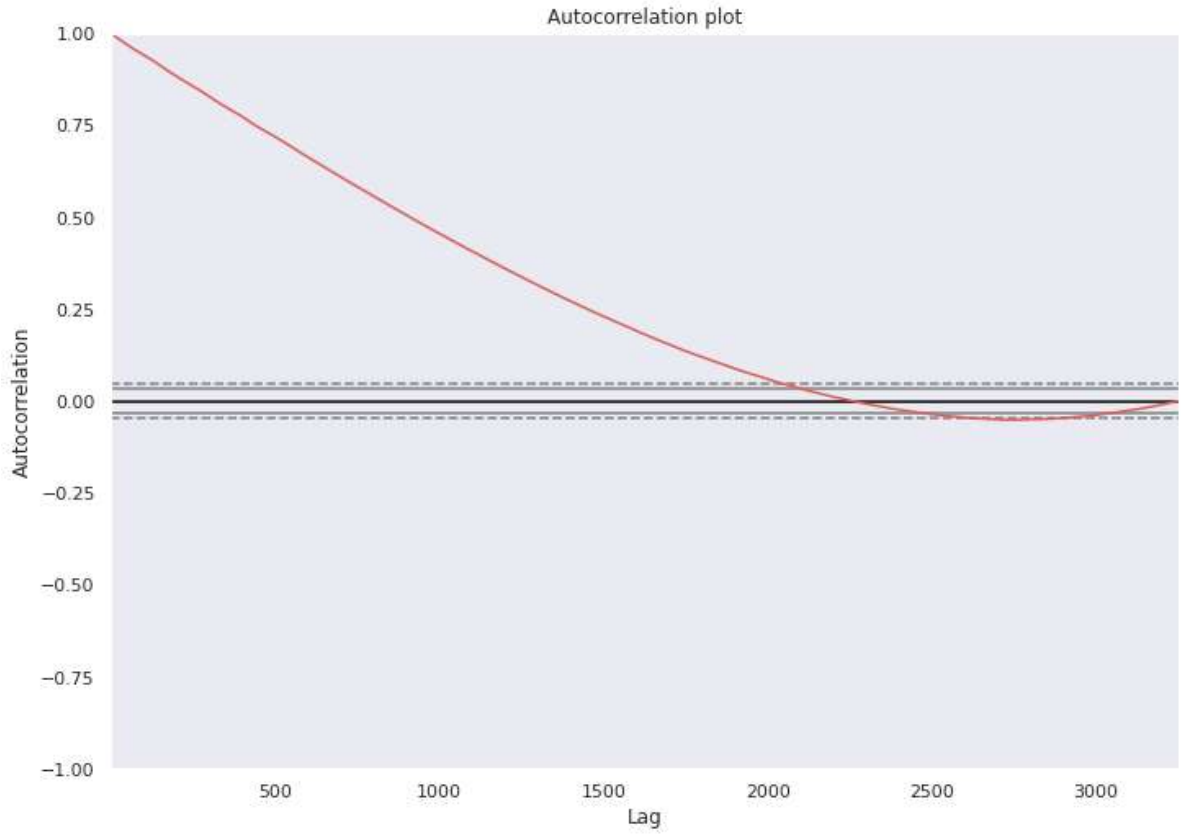
In [13]:

```
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 как параметр `validation_split` метода `model.fit()` .

Задание 3

Примените модель *ARIMA* для прогнозирования значений данного временного ряда.

In [15]:

```
! pip install pmdarima

! pip show pmdarima
```

```
Collecting pmdarima
  Downloading https://files.pythonhosted.org/packages/83/aa/feb76414043592c3
149059ab772a51de03fbf7544a8e19237f229a50a949/pmdarima-1.5.3-cp36-cp36m-manyl
inux1_x86_64.whl (https://files.pythonhosted.org/packages/83/aa/feb764140435
92c3149059ab772a51de03fbf7544a8e19237f229a50a949/pmdarima-1.5.3-cp36-cp36m-m
anylinux1_x86_64.whl) (1.5MB)
     |████████████████████████████████| 1.5MB 4.5MB/s
Requirement already satisfied: urllib3 in /usr/local/lib/python3.6/dist-pack
ages (from pmdarima) (1.24.3)
Requirement already satisfied: Cython>=0.29 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (0.29.16)
Requirement already satisfied: scipy>=1.3.2 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (1.4.1)
Requirement already satisfied: pandas>=0.19 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (1.0.3)
Requirement already satisfied: numpy>=1.17.3 in /usr/local/lib/python3.6/dis
t-packages (from pmdarima) (1.18.2)
Requirement already satisfied: joblib>=0.11 in /usr/local/lib/python3.6/dist
-packages (from pmdarima) (0.14.1)
Requirement already satisfied: scikit-learn>=0.22 in /usr/local/lib/python3.
6/dist-packages (from pmdarima) (0.22.2.post1)
Requirement already satisfied: statsmodels>=0.10.2 in /usr/local/lib/python
3.6/dist-packages (from pmdarima) (0.10.2)
Requirement already satisfied: pytz>=2017.2 in /usr/local/lib/python3.6/dist
-packages (from pandas>=0.19->pmdarima) (2018.9)
Requirement already satisfied: python-dateutil>=2.6.1 in /usr/local/lib/pyth
on3.6/dist-packages (from pandas>=0.19->pmdarima) (2.8.1)
Requirement already satisfied: patsy>=0.4.0 in /usr/local/lib/python3.6/dist
-packages (from statsmodels>=0.10.2->pmdarima) (0.5.1)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.6/dist-pac
kages (from python-dateutil>=2.6.1->pandas>=0.19->pmdarima) (1.12.0)
Installing collected packages: pmdarima
Successfully installed pmdarima-1.5.3
Name: pmdarima
Version: 1.5.3
Summary: Python's forecast::auto.arima equivalent
Home-page: http://alkaline-ml.com/pmdarima (http://alkaline-ml.com/pmdarima)
Author: None
Author-email: None
License: MIT
Location: /usr/local/lib/python3.6/dist-packages
Requires: scipy, pandas, statsmodels, Cython, numpy, scikit-learn, urllib3,
 joblib
Required-by:
```

In [0]:

```
test_period = 6 * 12
```

In [17]:

```
from pmdarima.arima import auto_arima

arima = auto_arima(all_df['Monthly Mean Total Sunspot Number'][:-test_period],
                   trace = True, error_action = 'ignore', suppress_warnings = True, seasona
```

```
Performing stepwise search to minimize aic
Fit ARIMA: (2, 0, 2)x(1, 0, 1, 12) (constant=True); AIC=29588.066, BIC=2963
6.583, Time=26.654 seconds
Fit ARIMA: (0, 0, 0)x(0, 0, 0, 12) (constant=True); AIC=35872.649, BIC=3588
4.778, Time=0.074 seconds
Fit ARIMA: (1, 0, 0)x(1, 0, 0, 12) (constant=True); AIC=30025.545, BIC=3004
9.804, Time=5.873 seconds
Fit ARIMA: (0, 0, 1)x(0, 0, 1, 12) (constant=True); AIC=32380.500, BIC=3240
4.758, Time=6.235 seconds
Fit ARIMA: (0, 0, 0)x(0, 0, 0, 12) (constant=False); AIC=38764.997, BIC=3877
1.062, Time=0.053 seconds
Fit ARIMA: (2, 0, 2)x(0, 0, 1, 12) (constant=True); AIC=29586.807, BIC=2962
9.260, Time=14.995 seconds
Fit ARIMA: (2, 0, 2)x(0, 0, 0, 12) (constant=True); AIC=29595.958, BIC=2963
2.345, Time=2.405 seconds
Fit ARIMA: (2, 0, 2)x(0, 0, 2, 12) (constant=True); AIC=29587.374, BIC=2963
5.891, Time=52.012 seconds
Fit ARIMA: (2, 0, 2)x(1, 0, 0, 12) (constant=True); AIC=29587.365, BIC=2962
9.817, Time=19.040 seconds
Fit ARIMA: (2, 0, 2)x(1, 0, 2, 12) (constant=True); AIC=29582.451, BIC=2963
7.033, Time=76.061 seconds
Fit ARIMA: (2, 0, 2)x(2, 0, 2, 12) (constant=True); AIC=29552.494, BIC=2961
3.140, Time=78.888 seconds
Near non-invertible roots for order (2, 0, 2)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.993)
Fit ARIMA: (2, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=29581.341, BIC=2963
5.923, Time=72.126 seconds
Fit ARIMA: (2, 0, 2)x(2, 0, 0, 12) (constant=True); AIC=29587.014, BIC=2963
5.531, Time=59.051 seconds
Fit ARIMA: (1, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=29579.106, BIC=2962
7.623, Time=48.393 seconds
Fit ARIMA: (1, 0, 2)x(1, 0, 1, 12) (constant=True); AIC=29586.021, BIC=2962
8.473, Time=20.851 seconds
Fit ARIMA: (1, 0, 2)x(2, 0, 0, 12) (constant=True); AIC=29585.061, BIC=2962
7.513, Time=43.642 seconds
Fit ARIMA: (1, 0, 2)x(2, 0, 2, 12) (constant=True); AIC=29551.544, BIC=2960
6.126, Time=80.763 seconds
Near non-invertible roots for order (1, 0, 2)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.995)
Fit ARIMA: (1, 0, 2)x(1, 0, 0, 12) (constant=True); AIC=29585.380, BIC=2962
1.768, Time=9.932 seconds
Fit ARIMA: (1, 0, 2)x(1, 0, 2, 12) (constant=True); AIC=29580.389, BIC=2962
8.906, Time=55.402 seconds
Fit ARIMA: (0, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=31382.620, BIC=3142
5.073, Time=35.907 seconds
Near non-invertible roots for order (0, 0, 2)(2, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
1.000)
Fit ARIMA: (1, 0, 1)x(2, 0, 1, 12) (constant=True); AIC=29629.733, BIC=2967
2.185, Time=59.111 seconds
Fit ARIMA: (1, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29580.861, BIC=2963
```

```
5.443, Time=54.306 seconds
Fit ARIMA: (0, 0, 1)x(2, 0, 1, 12) (constant=True); AIC=32043.176, BIC=3207
9.563, Time=30.535 seconds
Near non-invertible roots for order (0, 0, 1)(2, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
1.000)
Fit ARIMA: (0, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=31074.278, BIC=3112
2.795, Time=46.450 seconds
Near non-invertible roots for order (0, 0, 3)(2, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
1.000)
Fit ARIMA: (2, 0, 1)x(2, 0, 1, 12) (constant=True); AIC=29592.648, BIC=2964
1.165, Time=57.551 seconds
Fit ARIMA: (2, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29579.013, BIC=2963
9.659, Time=69.552 seconds
Fit ARIMA: (2, 0, 3)x(1, 0, 1, 12) (constant=True); AIC=29586.998, BIC=2964
1.579, Time=24.405 seconds
Fit ARIMA: (2, 0, 3)x(2, 0, 0, 12) (constant=True); AIC=29585.838, BIC=2964
0.420, Time=67.092 seconds
Fit ARIMA: (2, 0, 3)x(2, 0, 2, 12) (constant=True); AIC=29543.930, BIC=2961
0.641, Time=87.309 seconds
Near non-invertible roots for order (2, 0, 3)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.995)
Fit ARIMA: (2, 0, 3)x(1, 0, 0, 12) (constant=True); AIC=29586.591, BIC=2963
5.108, Time=18.751 seconds
Fit ARIMA: (2, 0, 3)x(1, 0, 2, 12) (constant=True); AIC=29580.547, BIC=2964
1.194, Time=78.227 seconds
Fit ARIMA: (3, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29578.087, BIC=2964
4.798, Time=89.062 seconds
Fit ARIMA: (3, 0, 3)x(1, 0, 1, 12) (constant=True); AIC=29586.766, BIC=2964
7.413, Time=25.645 seconds
Fit ARIMA: (3, 0, 3)x(2, 0, 0, 12) (constant=True); AIC=29584.745, BIC=2964
5.391, Time=87.351 seconds
Fit ARIMA: (3, 0, 3)x(2, 0, 2, 12) (constant=True); AIC=29546.099, BIC=2961
8.874, Time=91.555 seconds
Near non-invertible roots for order (3, 0, 3)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.995)
Fit ARIMA: (3, 0, 3)x(1, 0, 0, 12) (constant=True); AIC=29585.293, BIC=2963
9.875, Time=29.393 seconds
Fit ARIMA: (3, 0, 3)x(1, 0, 2, 12) (constant=True); AIC=29579.288, BIC=2964
5.999, Time=81.462 seconds
Fit ARIMA: (3, 0, 2)x(2, 0, 1, 12) (constant=True); AIC=29587.135, BIC=2964
7.781, Time=79.899 seconds
Fit ARIMA: (4, 0, 3)x(2, 0, 1, 12) (constant=True); AIC=29579.706, BIC=2965
2.482, Time=90.539 seconds
Fit ARIMA: (3, 0, 4)x(2, 0, 1, 12) (constant=True); AIC=29579.646, BIC=2965
2.421, Time=99.907 seconds
Fit ARIMA: (2, 0, 4)x(2, 0, 1, 12) (constant=True); AIC=29576.620, BIC=2964
3.331, Time=79.643 seconds
Fit ARIMA: (2, 0, 4)x(1, 0, 1, 12) (constant=True); AIC=29583.960, BIC=2964
4.607, Time=32.586 seconds
Fit ARIMA: (2, 0, 4)x(2, 0, 0, 12) (constant=True); AIC=29582.698, BIC=2964
3.345, Time=78.553 seconds
Fit ARIMA: (2, 0, 4)x(2, 0, 2, 12) (constant=True); AIC=29547.505, BIC=2962
0.281, Time=106.433 seconds
Near non-invertible roots for order (2, 0, 4)(2, 0, 2, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.994)
Fit ARIMA: (2, 0, 4)x(1, 0, 0, 12) (constant=True); AIC=29568.222, BIC=2962
```

```
2.804, Time=25.983 seconds
Fit ARIMA: (2, 0, 4)x(0, 0, 0, 12) (constant=True); AIC=29479.053, BIC=2952
7.570, Time=10.273 seconds
Fit ARIMA: (2, 0, 4)x(0, 0, 1, 12) (constant=True); AIC=29472.268, BIC=2952
6.849, Time=33.733 seconds
Fit ARIMA: (2, 0, 4)x(0, 0, 2, 12) (constant=True); AIC=29573.231, BIC=2963
3.878, Time=85.410 seconds
Fit ARIMA: (2, 0, 4)x(1, 0, 2, 12) (constant=True); AIC=29578.088, BIC=2964
4.799, Time=90.158 seconds
Fit ARIMA: (1, 0, 4)x(0, 0, 1, 12) (constant=True); AIC=29579.628, BIC=2962
8.145, Time=16.664 seconds
Fit ARIMA: (2, 0, 3)x(0, 0, 1, 12) (constant=True); AIC=29585.945, BIC=2963
4.462, Time=12.180 seconds
Fit ARIMA: (3, 0, 4)x(0, 0, 1, 12) (constant=True); AIC=29523.347, BIC=2958
3.993, Time=33.944 seconds
Fit ARIMA: (2, 0, 5)x(0, 0, 1, 12) (constant=True); AIC=29583.600, BIC=2964
4.247, Time=16.746 seconds
Fit ARIMA: (1, 0, 3)x(0, 0, 1, 12) (constant=True); AIC=29586.789, BIC=2962
9.241, Time=11.548 seconds
Fit ARIMA: (1, 0, 5)x(0, 0, 1, 12) (constant=True); AIC=29579.796, BIC=2963
4.378, Time=23.102 seconds
Fit ARIMA: (3, 0, 3)x(0, 0, 1, 12) (constant=True); AIC=29584.591, BIC=2963
9.173, Time=22.977 seconds
Fit ARIMA: (3, 0, 5)x(0, 0, 1, 12) (constant=True); AIC=29449.139, BIC=2951
5.850, Time=37.918 seconds
Near non-invertible roots for order (3, 0, 5)(0, 0, 1, 12); setting score to
inf (at least one inverse root too close to the border of the unit circle:
0.997)
Total fit time: 2694.409 seconds
```

In [0]:

```
arima_forecast = arima.predict(n_periods = test_period)
```

In [19]:

```
from sklearn.metrics import mean_squared_error

mean_squared_error(all_df['Monthly Mean Total Sunspot Number'][-test_period:], arima_foreca
```

Out[19]:

```
3196.9943474969787
```

Задание 4

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

Сначала нужно создать датасет из данных.

In [20]:

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

! pip show tensorflow-gpu
```

```
     |████████████████████████████████| 516.2MB 26kB/s
Name: tensorflow-gpu
Version: 2.2.0rc3
Summary: TensorFlow is an open source machine learning framework for everyon
e.
Home-page: https://www.tensorflow.org/ (https://www.tensorflow.org/)
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: /usr/local/lib/python3.6/dist-packages
Requires: keras-preprocessing, astunparse, termcolor, protobuf, numpy, tenso
rflow-estimator, absl-py, tensorboard, gast, google-pasta, h5py, six, grpci
o, wheel, scipy, opt-einsum, wrapt
Required-by:
```

In [0]:

```
TIME_STEPS = 100

TEST_PERIOD = 1000
```

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

    print( len(_X_ts))

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

    for i in range(samples_n_):

        X_[i] = _X_ts[i:(i + _time_steps)]

        y_[i] = _X_ts[(i + _time_steps)]

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

In [23]:

```
X, y = timeseries_to_dataset(all_df['Monthly Mean Total Sunspot Number'][:-TEST_PERIOD].val

X_test, y_test = timeseries_to_dataset(all_df['Monthly Mean Total Sunspot Number'][-TEST_PE
```

```
2252
1000
```

In [0]:

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

In [25]:

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

model = tf.keras.Sequential()

model.add(LSTM(8, activation = 'relu', return_sequences = True, input_shape = X.shape[-2:]))
model.add(LSTM(8, activation = 'relu'))
model.add(Dense(1))
```

```
WARNING:tensorflow:Layer lstm will not use cuDNN kernel since it doesn't mee
t the cuDNN kernel criteria. It will use generic GPU kernel as fallback when
running on GPU
WARNING:tensorflow:Layer lstm_1 will not use cuDNN kernel since it doesn't m
eet the cuDNN kernel criteria. It will use generic GPU kernel as fallback wh
en running on GPU
```

In [26]:

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

model.summary()
```

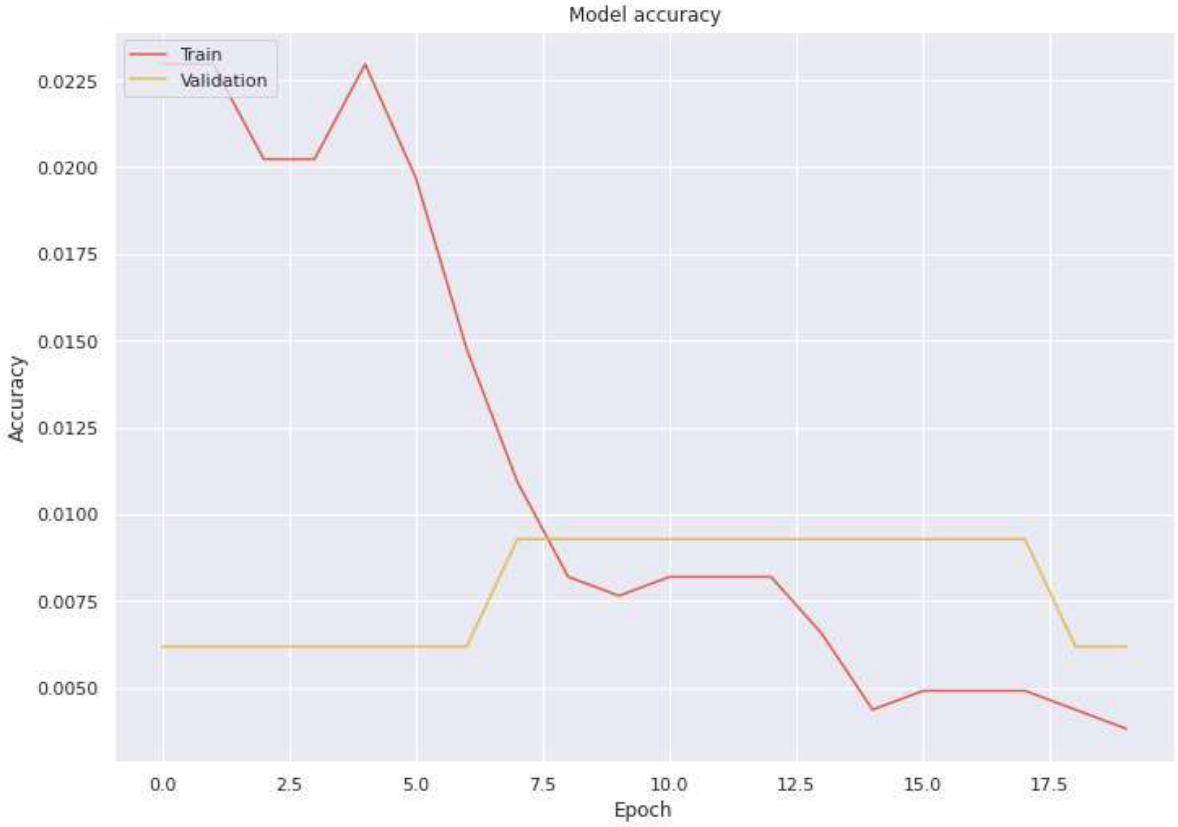
```
Model: "sequential"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
lstm (LSTM)                  (None, 100, 8)            320
_________________________________________________________________
lstm_1 (LSTM)                (None, 8)                 544
_________________________________________________________________
dense (Dense)                (None, 1)                 9
=================================================================
Total params: 873
Trainable params: 873
Non-trainable params: 0
_________________________________________________________________
```

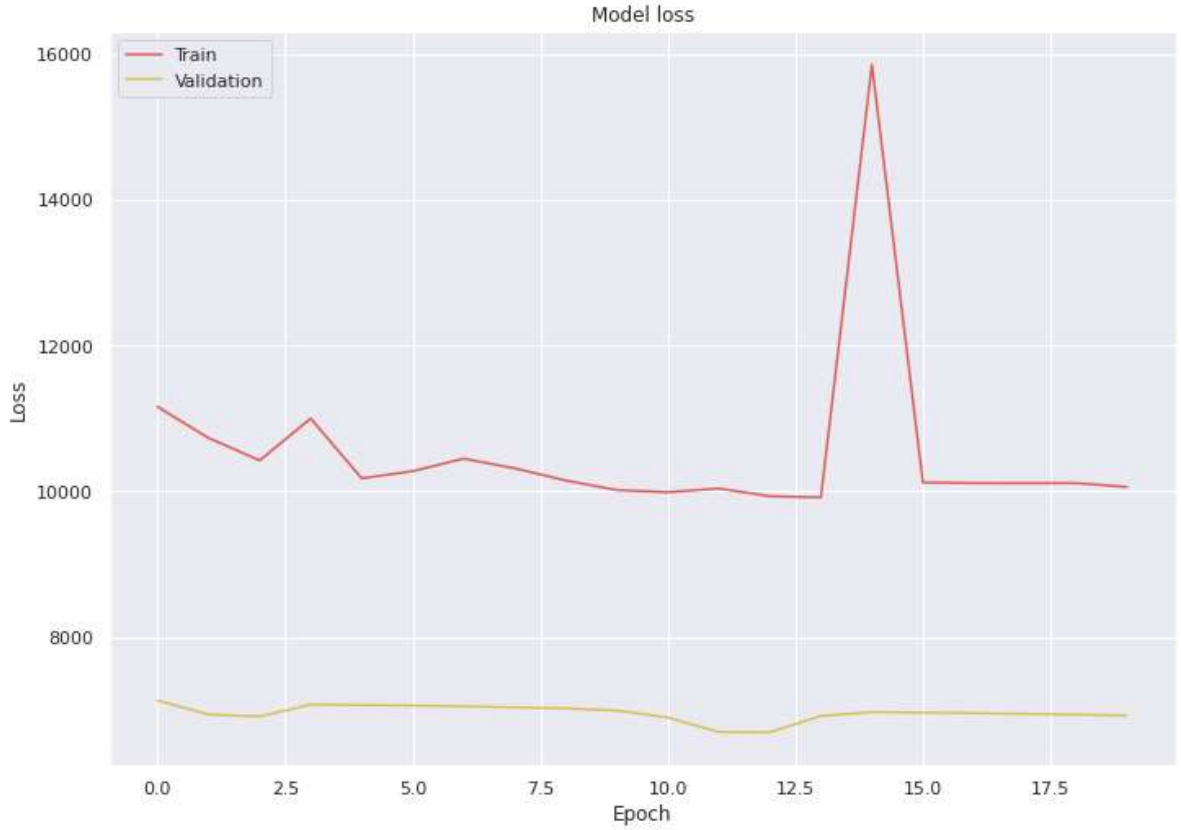
In [0]:

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

In [28]:

```
plt.plot(history.history['accuracy'])
plt.plot(history.history['val_accuracy'])
plt.title('Model accuracy')
plt.ylabel('Accuracy')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc = 'upper left')
plt.show()

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 = 'upper left')
plt.show()
```





In [29]:

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

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

```
29/29 [==============================] - 1s 28ms/step - loss: 14390.7432 - a
ccuracy: 0.0000e+00
Test mse, test accuracy: [14390.7431640625, 0.0]
```

Задание 5

Сравните качество прогноза моделей.

Какой максимальный результат удалось получить на контрольной выборке?

Нейронная сеть дала среднеквадратичную ошибку в 4 раза больше, чем ARIMA, а точность предсказания вообще равна нулю. Можно сделать вывод, что предсказание временных рядов требует более тонкой настройки архитектуры сетей.