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
```

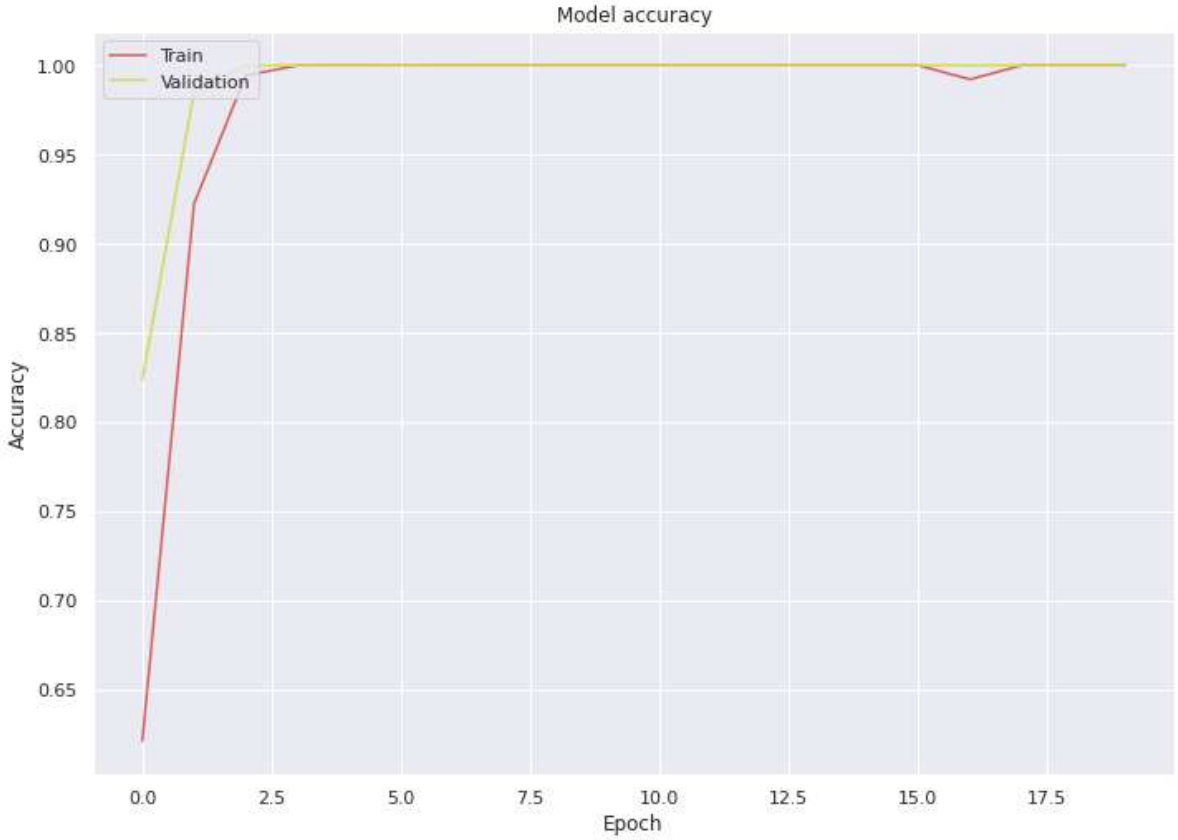
In [22]:

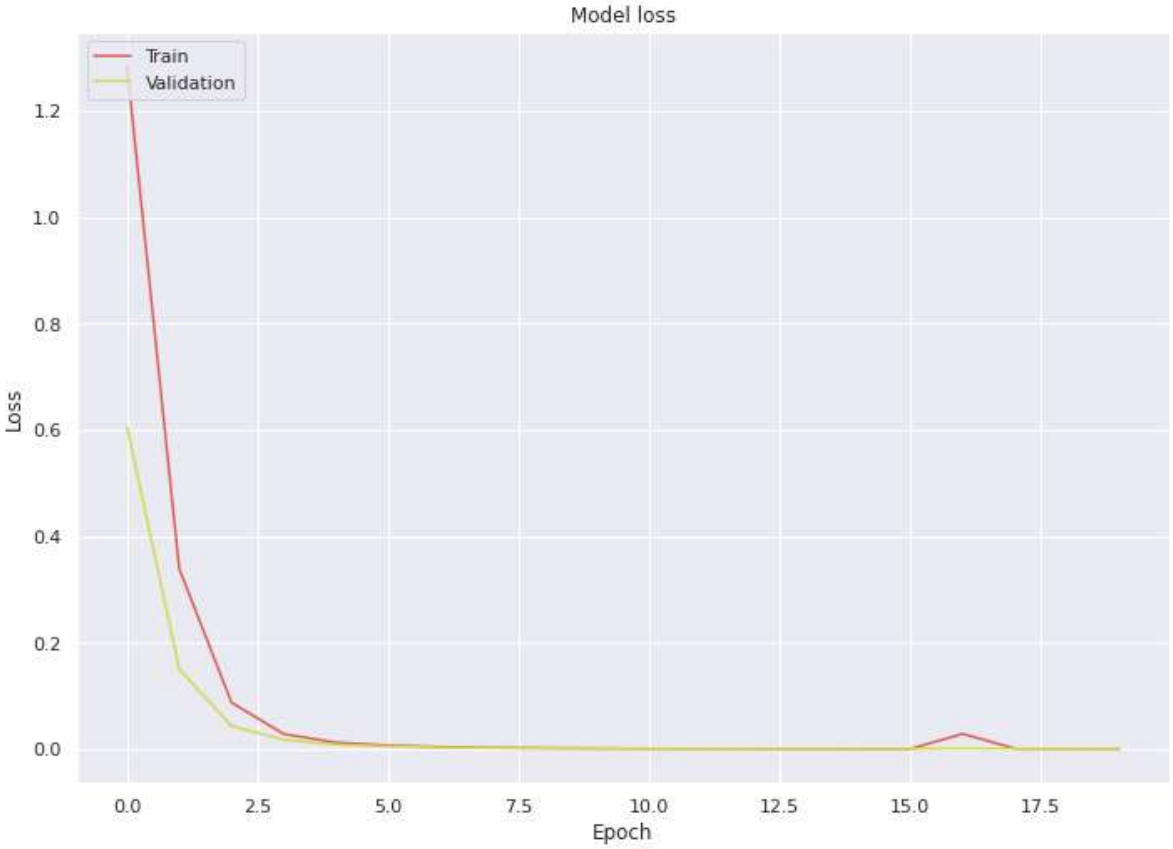
```
import seaborn as sns

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]:

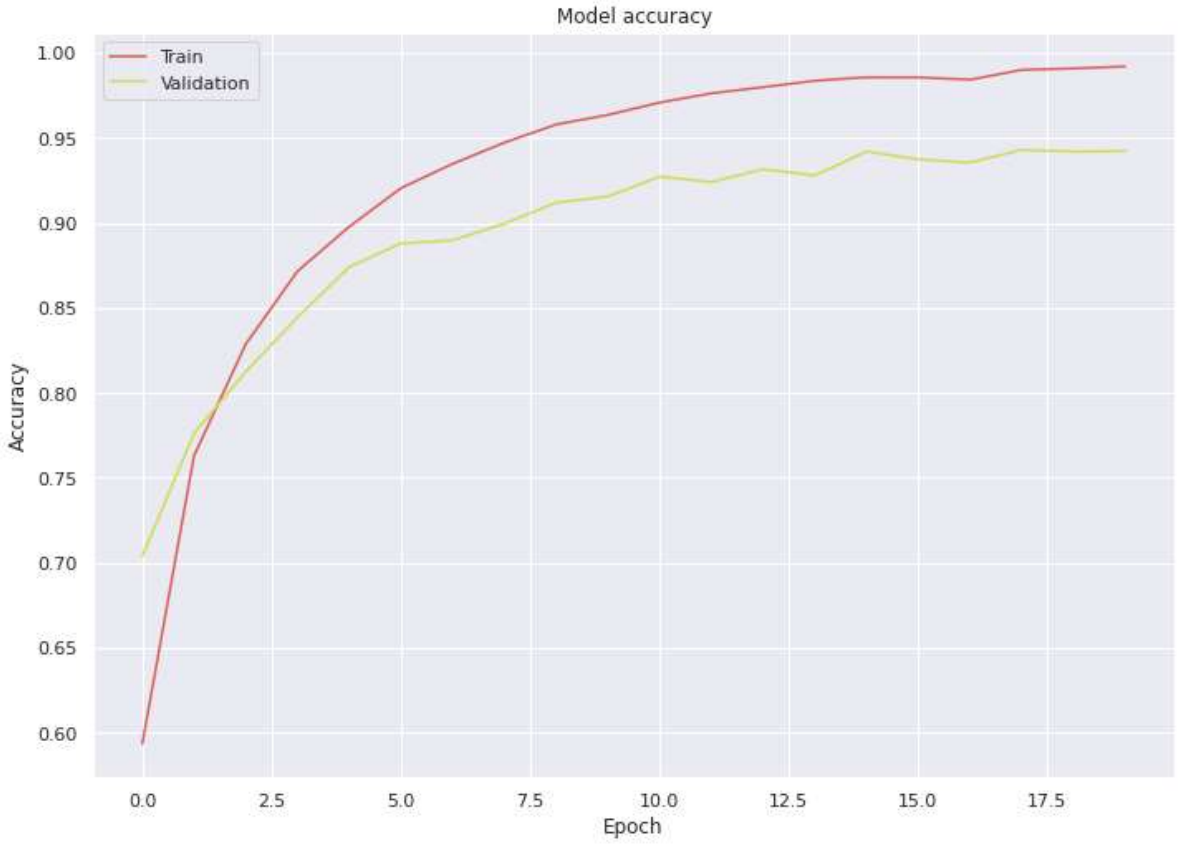
```
history_2 = model.fit(x = X_train_augmented, y = y_train_augmented, epochs = 20, validatior
```

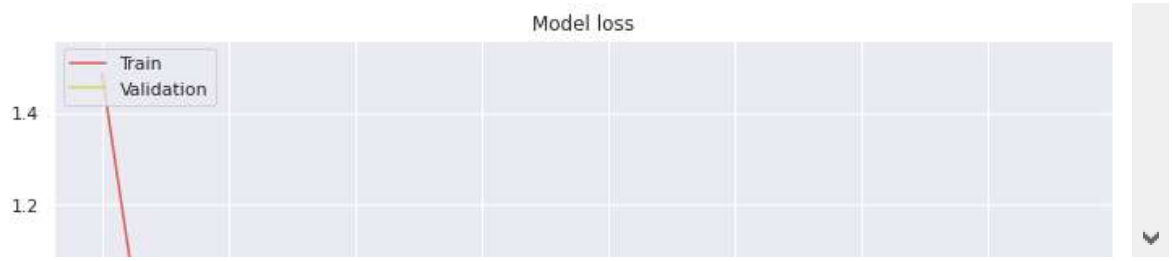
```
Epoch 1/20
730/730 [==============================] - 3s 4ms/step - loss: 1.4882 - cate
gorical_accuracy: 0.5935 - val_loss: 0.9350 - val_categorical_accuracy: 0.70
38
Epoch 2/20
730/730 [==============================] - 3s 4ms/step - loss: 0.7458 - cate
gorical_accuracy: 0.7629 - val_loss: 0.7156 - 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?