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
```

```
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: google-pasta, h5py, grpcio, opt-einsum, scipy, protobuf, wrapt, si
x, absl-py, tensorboard, wheel, gast, termcolor, numpy, tensorflow-estimato
r, keras-preprocessing, astunparse
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]:

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

```
Epoch 1/20
730/730 [==============================] - 2s 3ms/step - loss: 1.2958 - cate
gorical_accuracy: 0.6188 - val_loss: 0.5418 - val_categorical_accuracy: 0.86
57
Epoch 2/20
730/730 [==============================] - 2s 3ms/step - loss: 0.3189 - cate
gorical_accuracy: 0.9325 - val_loss: 0.1453 - val_categorical_accuracy: 0.98
30
Epoch 3/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0807 - cate
gorical_accuracy: 0.9949 - val_loss: 0.0389 - val_categorical_accuracy: 0.99
98
Epoch 4/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0262 - cate
gorical_accuracy: 0.9997 - val_loss: 0.0160 - val_categorical_accuracy: 0.99
98
Epoch 5/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0111 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0099 - val_categorical_accuracy: 0.99
98
Epoch 6/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0058 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0044 - val_categorical_accuracy: 1.00
00
Epoch 7/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0033 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0028 - val_categorical_accuracy: 1.00
00
Epoch 8/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0145 - cate
gorical_accuracy: 0.9964 - val_loss: 0.0043 - val_categorical_accuracy: 1.00
00
Epoch 9/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0020 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0014 - val_categorical_accuracy: 1.00
00
Epoch 10/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0011 - cate
gorical_accuracy: 1.0000 - val_loss: 9.1311e-04 - val_categorical_accuracy:
1.0000
Epoch 11/20
730/730 [==============================] - 2s 3ms/step - loss: 7.3560e-04 -
categorical_accuracy: 1.0000 - val_loss: 6.3469e-04 - val_categorical_accura
cy: 1.0000
Epoch 12/20
730/730 [==============================] - 2s 3ms/step - loss: 5.1847e-04 -
categorical_accuracy: 1.0000 - val_loss: 4.7602e-04 - val_categorical_accura
cy: 1.0000
Epoch 13/20
730/730 [==============================] - 2s 3ms/step - loss: 3.6735e-04 -
categorical_accuracy: 1.0000 - val_loss: 3.2560e-04 - val_categorical_accura
cy: 1.0000
Epoch 14/20
730/730 [==============================] - 2s 3ms/step - loss: 2.6058e-04 -
categorical_accuracy: 1.0000 - val_loss: 2.3029e-04 - val_categorical_accura
cy: 1.0000
Epoch 15/20
```

```
730/730 [==============================] - 2s 3ms/step - loss: 1.8485e-04 -
categorical_accuracy: 1.0000 - val_loss: 1.6227e-04 - val_categorical_accura
cy: 1.0000
Epoch 16/20
730/730 [==============================] - 2s 3ms/step - loss: 1.2857e-04 -
categorical_accuracy: 1.0000 - val_loss: 1.1164e-04 - val_categorical_accura
cy: 1.0000
Epoch 17/20
730/730 [==============================] - 2s 3ms/step - loss: 8.9278e-05 -
categorical_accuracy: 1.0000 - val_loss: 7.7735e-05 - val_categorical_accura
cy: 1.0000
Epoch 18/20
730/730 [==============================] - 2s 3ms/step - loss: 6.2117e-05 -
categorical_accuracy: 1.0000 - val_loss: 5.2588e-05 - val_categorical_accura
cy: 1.0000
Epoch 19/20
730/730 [==============================] - 2s 3ms/step - loss: 4.2049e-05 -
categorical_accuracy: 1.0000 - val_loss: 3.7133e-05 - val_categorical_accura
cy: 1.0000
Epoch 20/20
730/730 [==============================] - 2s 3ms/step - loss: 2.8740e-05 -
categorical_accuracy: 1.0000 - val_loss: 2.5821e-05 - val_categorical_accura
cy: 1.0000
```

Out[20]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd5b1433ba8>
```

In [21]:

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

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

```
225/225 [==============================] - 0s 2ms/step - loss: 0.8982 - cate
gorical_accuracy: 0.8256
Test loss, test accuracy: [0.8982465267181396, 0.8255716562271118]
```

За 20 эпох удалось достичь точности 82% 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 [23]:

```
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[23]:

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

In [0]:

```
y_train_augmented = y_train
```

In [25]:

```
model.fit(x = X_train_augmented, y = y_train_augmented, epochs = 20, validation_split = 0.1
```

```
Epoch 1/20
730/730 [==============================] - 2s 3ms/step - loss: 1.5178 - cate
gorical_accuracy: 0.5917 - val_loss: 0.9175 - val_categorical_accuracy: 0.72
42
Epoch 2/20
730/730 [==============================] - 2s 3ms/step - loss: 0.7474 - cate
gorical_accuracy: 0.7681 - val_loss: 0.6898 - val_categorical_accuracy: 0.79
17
Epoch 3/20
730/730 [==============================] - 2s 3ms/step - loss: 0.5514 - cate
gorical_accuracy: 0.8275 - val_loss: 0.6034 - val_categorical_accuracy: 0.81
26
Epoch 4/20
730/730 [==============================] - 2s 3ms/step - loss: 0.4323 - cate
gorical_accuracy: 0.8700 - val_loss: 0.5026 - val_categorical_accuracy: 0.84
41
Epoch 5/20
730/730 [==============================] - 2s 3ms/step - loss: 0.3474 - cate
gorical_accuracy: 0.8959 - val_loss: 0.4365 - val_categorical_accuracy: 0.86
67
Epoch 6/20
730/730 [==============================] - 2s 3ms/step - loss: 0.2858 - cate
gorical_accuracy: 0.9161 - val_loss: 0.3873 - val_categorical_accuracy: 0.88
23
Epoch 7/20
730/730 [==============================] - 2s 3ms/step - loss: 0.2393 - cate
gorical_accuracy: 0.9299 - val_loss: 0.3671 - val_categorical_accuracy: 0.88
66
Epoch 8/20
730/730 [==============================] - 2s 3ms/step - loss: 0.1912 - cate
gorical_accuracy: 0.9464 - val_loss: 0.3188 - val_categorical_accuracy: 0.90
22
Epoch 9/20
730/730 [==============================] - 2s 3ms/step - loss: 0.1655 - cate
gorical_accuracy: 0.9549 - val_loss: 0.3211 - val_categorical_accuracy: 0.90
00
Epoch 10/20
730/730 [==============================] - 2s 3ms/step - loss: 0.1389 - cate
gorical_accuracy: 0.9637 - val_loss: 0.2843 - val_categorical_accuracy: 0.91
53
Epoch 11/20
730/730 [==============================] - 2s 3ms/step - loss: 0.1186 - cate
gorical_accuracy: 0.9684 - val_loss: 0.2725 - val_categorical_accuracy: 0.91
45
Epoch 12/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0984 - cate
gorical_accuracy: 0.9751 - val_loss: 0.2554 - val_categorical_accuracy: 0.91
92
Epoch 13/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0848 - cate
gorical_accuracy: 0.9790 - val_loss: 0.2307 - val_categorical_accuracy: 0.92
72
Epoch 14/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0729 - cate
gorical_accuracy: 0.9834 - val_loss: 0.2121 - val_categorical_accuracy: 0.93
23
Epoch 15/20
```

```
730/730 [==============================] - 2s 3ms/step - loss: 0.0724 - cate
gorical_accuracy: 0.9811 - val_loss: 0.2220 - val_categorical_accuracy: 0.92
72
Epoch 16/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0627 - cate
gorical_accuracy: 0.9843 - val_loss: 0.2005 - val_categorical_accuracy: 0.93
98
Epoch 17/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0499 - cate
gorical_accuracy: 0.9891 - val_loss: 0.2056 - val_categorical_accuracy: 0.93
78
Epoch 18/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0469 - cate
gorical_accuracy: 0.9899 - val_loss: 0.2001 - val_categorical_accuracy: 0.93
66
Epoch 19/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0539 - cate
gorical_accuracy: 0.9859 - val_loss: 0.2164 - val_categorical_accuracy: 0.93
66
Epoch 20/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0478 - cate
gorical_accuracy: 0.9872 - val_loss: 0.1852 - val_categorical_accuracy: 0.94
34
```

Out[25]:

```
<tensorflow.python.keras.callbacks.History at 0x7fd58874f5c0>
```

In [26]:

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

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

```
225/225 [==============================] - 0s 2ms/step - loss: 0.2964 - cate
gorical_accuracy: 0.9140
Test loss, test accuracy: [0.296428382396698, 0.9139710068702698]
```

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?