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: keras-preprocessing, termcolor, absl-py, grpcio, wheel, scipy, ast
unparse, numpy, tensorflow-estimator, h5py, protobuf, opt-einsum, gast, six,
tensorboard, wrapt, google-pasta
Required-by:
```

In [0]:

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

X_train = np.asarray(list(train_df_reshaped['image']))
X_test = np.asarray(list(test_df_reshaped['image']))

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 [14]:

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

Out[14]:

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

In [0]:

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

In [0]:

```
import tensorflow as tf
```

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

```
Epoch 1/20
730/730 [==============================] - 3s 3ms/step - loss: 0.7144 - cate
gorical_accuracy: 0.8216 - val_loss: 0.1296 - val_categorical_accuracy: 0.98
57
Epoch 2/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0620 - cate
gorical_accuracy: 0.9943 - val_loss: 0.0209 - val_categorical_accuracy: 0.99
95
Epoch 3/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0136 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0083 - val_categorical_accuracy: 0.99
95
Epoch 4/20
730/730 [==============================] - 2s 3ms/step - loss: 0.1373 - cate
gorical_accuracy: 0.9634 - val_loss: 0.0284 - val_categorical_accuracy: 0.99
76
Epoch 5/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0107 - cate
gorical_accuracy: 0.9998 - val_loss: 0.0054 - val_categorical_accuracy: 1.00
00
Epoch 6/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0036 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0028 - val_categorical_accuracy: 1.00
00
Epoch 7/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0020 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0018 - val_categorical_accuracy: 1.00
00
Epoch 8/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0013 - cate
gorical_accuracy: 1.0000 - val_loss: 0.0013 - val_categorical_accuracy: 1.00
00
Epoch 9/20
730/730 [==============================] - 2s 3ms/step - loss: 8.6329e-04 -
categorical_accuracy: 1.0000 - val_loss: 8.2630e-04 - val_categorical_accura
cy: 1.0000
Epoch 10/20
730/730 [==============================] - 2s 3ms/step - loss: 5.9700e-04 -
categorical_accuracy: 1.0000 - val_loss: 5.6174e-04 - val_categorical_accura
cy: 1.0000
Epoch 11/20
730/730 [==============================] - 2s 3ms/step - loss: 3.8820e-04 -
categorical_accuracy: 1.0000 - val_loss: 3.7687e-04 - val_categorical_accura
cy: 1.0000
Epoch 12/20
730/730 [==============================] - 2s 3ms/step - loss: 2.6361e-04 -
categorical_accuracy: 1.0000 - val_loss: 2.7043e-04 - val_categorical_accura
cy: 1.0000
Epoch 13/20
730/730 [==============================] - 2s 3ms/step - loss: 1.8208e-04 -
categorical_accuracy: 1.0000 - val_loss: 1.9701e-04 - val_categorical_accura
cy: 1.0000
Epoch 14/20
730/730 [==============================] - 2s 3ms/step - loss: 1.2523e-04 -
categorical_accuracy: 1.0000 - val_loss: 1.3436e-04 - val_categorical_accura
cy: 1.0000
Epoch 15/20
```

```
730/730 [==============================] - 2s 3ms/step - loss: 8.3785e-05 -
categorical_accuracy: 1.0000 - val_loss: 9.1194e-05 - val_categorical_accura
cy: 1.0000
Epoch 16/20
730/730 [==============================] - 2s 3ms/step - loss: 5.6255e-05 -
categorical_accuracy: 1.0000 - val_loss: 8.4611e-05 - val_categorical_accura
cy: 1.0000
Epoch 17/20
730/730 [==============================] - 2s 3ms/step - loss: 3.8880e-05 -
categorical_accuracy: 1.0000 - val_loss: 4.7102e-05 - val_categorical_accura
cy: 1.0000
Epoch 18/20
730/730 [==============================] - 2s 3ms/step - loss: 0.2848 - cate
gorical_accuracy: 0.9219 - val_loss: 0.1512 - val_categorical_accuracy: 0.96
41
Epoch 19/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0610 - cate
gorical_accuracy: 0.9873 - val_loss: 0.0155 - val_categorical_accuracy: 0.99
95
Epoch 20/20
730/730 [==============================] - 2s 3ms/step - loss: 0.0104 - cate
gorical_accuracy: 0.9992 - val_loss: 0.0050 - val_categorical_accuracy: 0.99
95
```

Out[19]:

```
<tensorflow.python.keras.callbacks.History at 0x7f51b04778d0>
```

In [20]:

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

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

```
225/225 [==============================] - 0s 2ms/step - loss: 0.3798 - cate
gorical_accuracy: 0.8755
Test loss, test accuracy: [0.37977278232574463, 0.8754879832267761]
```

За 20 эпох удалось достичь точности в 87% на тестовой выборке.

Задание 3

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

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?