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, six, google-pasta, tensorflow-estimator, gast, tensorb
oard, grpcio, scipy, termcolor, wrapt, keras-preprocessing, protobuf, absl-p
y, numpy, h5py, opt-einsum, wheel
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)
```

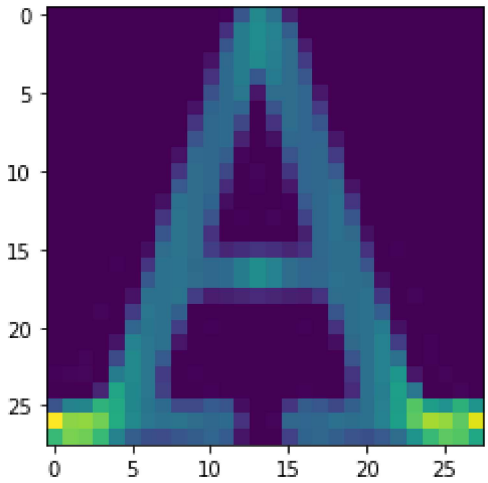
In [11]:

```
import matplotlib.pyplot as plt

plt.imshow(x[100].squeeze())
```

Out[11]:

```
<matplotlib.image.AxesImage at 0x7f6ba8e81a90>
```



In [0]:

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

In [13]:

```
from tensorflow.keras.utils import to_categorical

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

y.shape
```

Out[13]:

```
(415751, 10)
```

In [14]:

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

y_test.shape
```

Out[14]:

```
(46195, 10)
```

In [0]:

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

In [0]:

```
DENSE_LAYER_WIDTH = 5000
```

In [17]:

```
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 [19]:

```
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 [22]:

```
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 [==============================] - 114s 304us/sample - loss:
0.4815 - categorical_accuracy: 0.8573 - val_loss: 2.9091 - val_categorical_a
ccuracy: 0.3111
Epoch 2/10
374175/374175 [==============================] - 114s 304us/sample - loss:
0.3455 - categorical_accuracy: 0.8956 - val_loss: 2.7738 - val_categorical_a
ccuracy: 0.2886
Epoch 3/10
374175/374175 [==============================] - 113s 303us/sample - loss:
0.2926 - categorical_accuracy: 0.9110 - val_loss: 1.9089 - val_categorical_a
ccuracy: 0.5299
Epoch 4/10
374175/374175 [==============================] - 113s 302us/sample - loss:
0.2517 - categorical_accuracy: 0.9231 - val_loss: 2.2537 - val_categorical_a
ccuracy: 0.4842
Epoch 5/10
374175/374175 [==============================] - 114s 303us/sample - loss:
0.2144 - categorical_accuracy: 0.9342 - val_loss: 2.7603 - val_categorical_a
ccuracy: 0.3479
Epoch 6/10
374175/374175 [==============================] - 113s 301us/sample - loss:
0.1803 - categorical_accuracy: 0.9447 - val_loss: 2.1086 - val_categorical_a
ccuracy: 0.5454
Epoch 7/10
374175/374175 [==============================] - 114s 305us/sample - loss:
0.1466 - categorical_accuracy: 0.9551 - val_loss: 2.4950 - val_categorical_a
ccuracy: 0.4780
Epoch 8/10
374175/374175 [==============================] - 113s 302us/sample - loss:
0.1170 - categorical_accuracy: 0.9647 - val_loss: 2.0592 - val_categorical_a
ccuracy: 0.6014
Epoch 9/10
374175/374175 [==============================] - 113s 301us/sample - loss:
0.0925 - categorical_accuracy: 0.9728 - val_loss: 2.6105 - val_categorical_a
ccuracy: 0.5198
Epoch 10/10
374175/374175 [==============================] - 113s 302us/sample - loss:
0.0738 - categorical_accuracy: 0.9786 - val_loss: 2.7623 - val_categorical_a
ccuracy: 0.5034
```

Out[22]:

```
<tensorflow.python.keras.callbacks.History at 0x7f6ba8e81cf8>
```

In [23]:

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

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

```
Test loss, test accuracy: [0.5621323866975915, 0.88420826]
```

Лучшая точность построенной модели на тестовой выборке составила 88%.

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 [26]:

```
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 [27]:

```
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 [==============================] - 36s 97us/sample - loss: 0.5
802 - categorical_accuracy: 0.8338 - val_loss: 3.2642 - val_categorical_accu
racy: 0.0382
Epoch 2/10
374175/374175 [==============================] - 36s 96us/sample - loss: 0.4
135 - categorical_accuracy: 0.8762 - val_loss: 3.0404 - val_categorical_accu
racy: 0.1671
Epoch 3/10
374175/374175 [==============================] - 36s 97us/sample - loss: 0.3
677 - categorical_accuracy: 0.8891 - val_loss: 2.6523 - val_categorical_accu
racy: 0.3309
Epoch 4/10
374175/374175 [==============================] - 36s 95us/sample - loss: 0.3
384 - categorical_accuracy: 0.8975 - val_loss: 2.3369 - val_categorical_accu
racy: 0.4434
Epoch 5/10
374175/374175 [==============================] - 36s 96us/sample - loss: 0.3
159 - categorical_accuracy: 0.9037 - val_loss: 2.4419 - val_categorical_accu
racy: 0.4272
Epoch 6/10
374175/374175 [==============================] - 36s 95us/sample - loss: 0.2
965 - categorical_accuracy: 0.9099 - val_loss: 2.2025 - val_categorical_accu
racy: 0.4449
Epoch 7/10
374175/374175 [==============================] - 36s 96us/sample - loss: 0.2
797 - categorical_accuracy: 0.9147 - val_loss: 2.2130 - val_categorical_accu
racy: 0.4772
Epoch 8/10
374175/374175 [==============================] - 37s 98us/sample - loss: 0.2
644 - categorical_accuracy: 0.9193 - val_loss: 2.1537 - val_categorical_accu
racy: 0.5218
Epoch 9/10
374175/374175 [==============================] - 36s 95us/sample - loss: 0.2
509 - categorical_accuracy: 0.9233 - val_loss: 1.8914 - val_categorical_accu
racy: 0.5983
Epoch 10/10
374175/374175 [==============================] - 36s 96us/sample - loss: 0.2
375 - categorical_accuracy: 0.9274 - val_loss: 2.2922 - val_categorical_accu
racy: 0.5274
```

Out[27]:

```
<tensorflow.python.keras.callbacks.History at 0x7f6ba828da58>
```

In [28]:

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

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

```
Test loss, test accuracy: [0.5017564680594646, 0.87370926]
```

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 [31]:

```
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 [==============================] - 30s 79us/sample - loss: 0.4
504 - categorical_accuracy: 0.8640 - val_loss: 2.7422 - val_categorical_accu
racy: 0.2516
Epoch 2/10
374175/374175 [==============================] - 30s 80us/sample - loss: 0.3
514 - categorical_accuracy: 0.8912 - val_loss: 2.9686 - val_categorical_accu
racy: 0.2343
Epoch 3/10
374175/374175 [==============================] - 29s 79us/sample - loss: 0.3
239 - categorical_accuracy: 0.8995 - val_loss: 2.3722 - val_categorical_accu
racy: 0.4675
Epoch 4/10
374175/374175 [==============================] - 30s 79us/sample - loss: 0.3
086 - categorical_accuracy: 0.9037 - val_loss: 2.9093 - val_categorical_accu
racy: 0.3062
Epoch 5/10
374175/374175 [==============================] - 30s 79us/sample - loss: 0.2
972 - categorical_accuracy: 0.9069 - val_loss: 2.3950 - val_categorical_accu
racy: 0.4489
Epoch 6/10
374175/374175 [==============================] - 30s 79us/sample - loss: 0.2
891 - categorical_accuracy: 0.9092 - val_loss: 2.7761 - val_categorical_accu
racy: 0.2896
Epoch 7/10
374175/374175 [==============================] - 30s 80us/sample - loss: 0.2
811 - categorical_accuracy: 0.9117 - val_loss: 2.7471 - val_categorical_accu
racy: 0.3655
Epoch 8/10
374175/374175 [==============================] - 30s 80us/sample - loss: 0.2
749 - categorical_accuracy: 0.9136 - val_loss: 1.9750 - val_categorical_accu
racy: 0.5368
Epoch 9/10
374175/374175 [==============================] - 29s 79us/sample - loss: 0.2
707 - categorical_accuracy: 0.9148 - val_loss: 2.1891 - val_categorical_accu
racy: 0.5343
Epoch 10/10
374175/374175 [==============================] - 30s 80us/sample - loss: 0.2
659 - categorical_accuracy: 0.9163 - val_loss: 2.2132 - val_categorical_accu
racy: 0.5457
```

Out[31]:

```
<tensorflow.python.keras.callbacks.History at 0x7f6ba7593048>
```

In [32]:

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

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

```
Test loss, test accuracy: [0.49347359862197515, 0.8702024]
```

Удивительно, но *LeNet-5* показала результат хуже, чем первая модель — 87% 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 — 72%;
    - с адапливны шагом — 72%;
- модель с двумя свёрточными слоями и одним полносвязным — 88%;
- модель с одним свёрточным слоем, операцией пулинга и одним полносвязным — 87%;
- *LeNet-5* — два свёрточных слоя две операции пулинга два полносвязных слоя — 87%.

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