Министерство образования и науки Российской Федерации

Федеральное государственное автономное образовательное учреждение высшего образования «Национальный исследовательский

Нижегородский государственный университет им. Н.И. Лобачевского»

Институт информационных технологий, математики и механики

Кафедра Математического обеспечения и суперкомпьютерных технологий

Отчёт по учебной практике

Применение библиотеки Caffe для решения задач семантической сегментации изображений из базы данных Cityscapes

**Выполнил:** студент группы 381406-1

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Доронин М.А.

Подпись

**Научный руководитель:**

к.т.н., ст.преп. каф. МОСТ ИИТММ  
\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_ Кустикова В.Д.

Подпись

Нижний Новгород

2017

Содержание

[Введение 3](#_Toc508969198)

[1 Постановка задачи 4](#_Toc508969199)

[2 Обзор библиотеки глубокого обучения и ее зависимостей 5](#_Toc508969200)

[2.1 Caffe 5](#_Toc508969201)

[2.2 Необходимые зависимости для установки Caffe 6](#_Toc508969202)

[3 Реализация скриптовых сценариев 9](#_Toc508969203)

[3.1 Создание файловой иерархии 9](#_Toc508969204)

[3.2 Загрузка и распаковка исходных кодов 9](#_Toc508969205)

[3.3 Экспорт системных переменных 10](#_Toc508969206)

[3.4 Сборка и установка зависимостей 10](#_Toc508969207)

[3.5 Сборка библиотеки Caffe 10](#_Toc508969208)

[4 Тестовые запуски 11](#_Toc508969209)

[5 Процесс работы с библиотекой Caffe 13](#_Toc508969210)

[5.1 Набор тренировочных и тестовых данных 13](#_Toc508969211)

[5.2 Преобразование набора данных к hdf5 формату 14](#_Toc508969212)

[5.3 Обучение нейронной сети 14](#_Toc508969213)

[Заключение 15](#_Toc508969214)

[Литература 16](#_Toc508969215)

[Ссылки 17](#_Toc508969216)

[Приложения 18](#_Toc508969217)

[1. Обучение сети LeNetна базе данных MNIST 18](#_Toc508969218)

[2. Создание списка изображений из базы данных 19](#_Toc508969219)

[3. Создание hdf5 базы данных 20](#_Toc508969220)

# Введение

Машинное обучение – класс методов искусственного интеллекта, которые характеризуются не прямым решением задачи, а обучаемостью в процессе применения решений для других схожих задач. Для построения таких методов используются средства из теории вероятности и математической статистики, численных методов и методов оптимизации, теории графов и других математических наук.

За счет многообразия и сложности применяемых научных знаний возникает проблема в реализации методов машинного обучения одним человеком: это достаточно трудоемко и ресурсоемко по времени. Поэтому каждый разработчик и проектировщик использует в своей работе знания, накопленные человечеством, а в данном случае эти знания представляют собой готовые библиотеки машинного обучения. Нетрудно догадаться, что разработчики этих библиотек также использовали чьи-то наработки для упрощения своей работы. А иногда без сторонних зависимостей в принципе невозможна разработка на той или иной архитектуре, системе или устройствах. Так, например, библиотека машинного обучения Caffe имеет зависимости от библиотек CUDA, BLAS, Boost, Protobuf и других.

Если разработчик использует персональный компьютер, то сборка и установка этих зависимостей зачастую ограничивается использованием встроенных консольных инструментов, таких как apt-get или pip install. Ровно таким же способом можно устанавливать и необходимые библиотеки машинного обучения. Однако, если перед программистом стоит цель обучить метод на более мощной вычислительной машине, то он может столкнуться с проблемой отсутствия прав администратора на кластере или сервере, с использованием которых он собирается решать поставленную задачу. Тогда появляется необходимость в сборке зависимостей и библиотек из исходных кодов и в их изолированной установке и использовании.

В данной работе будут рассмотрен вариант сборки библиотеки Caffe и её зависимостей, написаны скриптовые сценарии для загрузки, распаковки, сборки и установки этих библиотек. А также протестирована работоспособность на пробных запусках обучения и тестирования нейронной сети.

Во второй части работы будет изучена структура набора тренировочных данных и разметки Cityscapes для решения задачи семантической сегментации дорожных сцен. А также выполнено преобразование набора данных и разметки в формат, доступный библиотеке Caffe. Следующим шагом будут выполнено обучение глубокой нейронной сети на этих данных.

# Постановка задачи

В ходе учебной практики требуется выполнить следующие задачи:

1. Провести краткий обзор выбранной библиотеки машинного обучения и зависимых библиотек.
2. Разработать скриптовые сценарии для загрузки и распаковки исходных кодов библиотек, для сборки этих библиотек и их установки.
3. Провести тестовые запуски на данных базы MNIST [16].
4. Изучить структуру набора данных и разметки базы CityScapes для решения задачи семантической сегментации дорожных сцен. Преобразование набора данных и разметки CityScapes в формат, доступный библиотеке Caffe.
5. Провести обучение глубокой нейронной сети на примере существующих моделей сетей.

По результатам выполненной работы необходимо предоставить отчёт с вычислительными результатами и с ссылками на репозиторий с реализованным функционалом.

# Обзор библиотеки глубокого обучения и ее зависимостей

В [2] рассмотрены несколько библиотек глубокого обучения, среди которых выбрана библиотека Tensorflow для тестовых запусков как наиболее доступная на ОС Windows. Однако специфика выбранной в той же работе задачи, а именно построение нейронной сети для семантической сегментации изображений дорожных сцен, состоит в том, что для обучения подобных сетей требуются куда большие вычислительные мощности, в отличие от мощностей персонального компьютера. Поэтому было принято решение использовать вычислительные узлы Linux сегмента кластера “Лобачевский” и библиотеку Caffe, как наиболее распространенную на ОС Linux.

## Caffe

В данном разделе приведен краткий обзор, являющийся выдержкой из работы [2], библиотеки машинного обучения Caffe [4] и ее достоинства.

Библиотека Caffe – фреймворк глубокого обучения с открытым исходным кодом на языке С++. При этом продукт полностью поддерживает написание пользовательским алгоритмов на Python и NumPy, а также совместим с MATLAB. Caffe предлагает широкий инструментарий для создания и применения современных алгоритмов глубокого обучения.

Достоинства библиотеки Caffe:

* «Чистая» архитектура, которая позволяет осуществить мгновенное развертывание. Сети определяются простыми файлами конфигурации, благодаря чему переключение между центральным и графическим процессором происходит легко. Для ускорения вычислений Caffe может быть запущена на GPU с использованием базовых возможностей технологии CUDA или библиотеки примитивов глубокого обучения cuDNN.
* Открытый код позволяет не только контролировать внедрение, но и модифицировать его под свои нужды. В первые 6 месяцев активного использования Caffe более 300 независимых разработчиков внесли свой вклад в развитие библиотеки, просто подстраивая ее под себя.
* На данный момент Caffe лидирует по скорости обучения среди многих библиотек, реализующих сверточные сети.
* Еще одно преимущество Caffe – наличие уже готовых свободно распространяемых моделей обучения, полностью интегрированных в систему.

## Необходимые зависимости для установки Caffe

Библиотека Caffe является сложным программным продуктом, который использует множество различных технологий. Поэтому для её работы требуется установка сторонних библиотек. Все зависимости кратко рассмотрим в данном разделе.

CUDA

CUDA [5] – программно-аппаратная архитектура для проведения параллельных вычислений, увеличивающая вычислительную производительность за счет использования графических процессоров фирмы Nvidia. CUDA SDK предоставляет возможность реализации на С-подобном языке алгоритмов, которые будут исполняться на графических процессорах. Архитектура CUDA даёт разработчику возможности организовывать доступ к набору инструкций графического ускорителя и управлять его памятью самостоятельно.

Для работы Caffe в режиме GPU необходима установка CUDA.

BLAS

BLAS (англ. Basic Linear Algebra Subprograms) – базовые подпрограммы линейной алгебры, стандарт де-факто интерфейса программирования приложений для создания программ и библиотек, в которых выполняются основные операции линейной алгебры, такие как умножения векторов и матриц. Впервые стандарт был опубликован в 1979 году и использован для создания крупных программных пакетов, например, LAPACK. Различные производители заинтересованы в создании высокооптимизированных реализация интерфейса BLAS для увеличения производительности своих систем. Например, MKL – реализация компании Intel, ATLAS – переносимый самооптимизирующийся BLAS.

Данный интерфейс используется библиотекой Caffe по очевидным причинам. В алгоритмах машинного обучения достаточно много векторных и матричных операций.

В данной работе используется библиотека ATLAS [6].

Boost

Boost [7] – кроссплатформенный высокоуровневый интерфейс, представляющий собой собрание библиотек классов на языке С++, для решения повседневных подзадач программирования. Boost направлен на исследования и расширяемость функциональности (метапрограммирование, обобщенное программирование с активным использованием шаблонов), является своего рода “испытательным полигоном”, на котором тестируются некоторые функциональности, являющиеся кандидатами на включение в стандарт языка.

Библиотека Boost используется разработчиками Caffe и также необходима для её работы.

Protobuf

Protocol Buffers [8] – протокол сериализации структурированных данных, предложенный компанией Google как эффективная бинарная альтернатива текстовому формату XML. Разработчики утверждают, что protobuf проще, компактнее и быстрее, чем XML, так как он осуществляет передачу бинарных данных, оптимизированных под минимальный размер сообщения. По замыслу разработчиков, сначала должна быть описана структура данных, которая затем компилируется в классы. Вместе с классами идет код их сериализации компактном формате представления. Чтение и запись данных доступны в высокоуровневых языках программирования, таких как Java, C++ или Python.

Protobuf используется библиотекой Caffe, например, для описания структуры и архитектуры нейронных сетей.

Glog

Данная библиотека [9] является реализацией интерфейса логирования от компании Google. Использует API языка С++ и поддерживает макросы и потоковые ввод/вывод.

Gflags

Gflags [10] – программный пакет, содержащий библиотеку, написанную на языке С++, реализующую обработку аргументов и параметров командной строки. Она включает поддержку стандартных типов данных, таких как строки, и предоставляет возможность определения флагов и параметров в исходных файлах, в которых они используются.

HDF5

Hierarchical Data Format [11] – название формата файлов, разработанного для хранения большого объема цифровой информации. Соответственно, библиотека Caffe требует установку библиотеки hdf5 для работы с данным форматом. Используется он в основном для структуризации и хранения обучающих и тестовых данных.

OpenCV

OpenCV [12] (англ. Open Source Computer Vision Library, библиотека компьютерного зрения с открытым исходным кодом) – библиотека алгоритмов компьютерного зрения, обработки изображений и численных алгоритмов общего назначения с открытым кодом. Реализована на C/C++, также разрабатывается для Python, Java, Ruby, Matlab, Lua и других языков.

OpenCV и Caffe используются совместно для решения задач компьютерного зрения методами глубокого обучения.

LMDB и LevelDB

Lightning Memory-Mapped Database (LMDB) [13] – это библиотека программного обеспечения, которая обеспечивает высокопроизводительные транзакционные операции с базами данных. LMDB написан на C с привязками API для нескольких языков программирования. LMDB также может использоваться одновременно в многопоточной или многопроцессорной среде с линейным масштабированием производительности.

LevelDB [14] – это высокопроизводительная NoSQL система для хранения данных в формате ключ/значение, разработанная компанией Google. Хранилище LevelDB написано на языке С++ и подключается к приложениям в виде разделяемой библиотеки, обеспечивая возможность хранения упорядоченных наборов данных, в которых строковые ключи сопоставлены со строковыми значениями.

Caffe может использовать входные данные в формате lmdb или leveldb.

CMake

CMake [15] – это кроссплатформенная система автоматизации сборки ПО из исходных кодов. CMake не занимается непосредственно сборкой или компиляцией, а лишь генерирует файлы управления сборкой из файлов CMakeLists.txt. В случае Unix-систем, CMake генерирует Make-файлы для сборки с помощью утилиты make.

В данной работе CMake используется для удобства сборки некоторых из зависимостей. Также с помощью этой утилиты можно собрать библиотеку Caffe, но в данной работе рассмотрен случай сборки непосредственно из Makefile.

# Реализация скриптовых сценариев

Для написания скриптов был выбран синтаксис bash как широко известный командный интерпретатор в Unix-подобных системах, в особенности в GNU/Linux.

Точкой входа для начала работы реализованных скриптов для установки библиотек является файл install-all.sh [3].

## Создание файловой иерархии

После применения скриптов подразумевается следующее представление в файловой системе:

* Root-директория, откуда запускался корневой скрипт install-all.sh.
  + caffe – директория с исходным кодом библиотеки Caffe.
    - build – директория с артефактами сборки.
  + data – директория с данными для обучения.
  + deps – директория с библиотеками-зависимостями.
    - bin – директория с исполняемыми файлами, полученными после сборки зависимостей.
    - doc – директория с документацией к библиотекам.
    - include – директория с заголовочными файлами, полученными после установки зависимостей.
    - lib – директория, содержащая разделяемые библиотеки, полученные в результате компиляции зависимостей. Необходимы для линковки при использовании в других программах и для работы исполняемых файлов из bin/.
    - lib64 – то же, только для архитектуры x64.
    - share – общие данные установленных программ.
    - src – исходные коды библиотек. Получены в результате распаковки загруженных архивов.
    - zips – загруженные архивы с исходными кодами.

Создание данной структуры происходит за счет использования команды mkdir, либо автоматически при исполнении make, make install или git clone.

## Загрузка и распаковка исходных кодов

Для загрузки архивов с сайтов разработчиков используется утилита wget. Для распаковки архивов либо утилита tar, либо unzip, в зависимости от типа файла. Реализация загрузки и распаковки в bash представлена в файле download-and-unzip-deps.sh. Некоторые исходные файлы загружаются непосредственно из официальных репозиториев командой git clone.

## ***Экспорт системных переменных***

Для корректной работы компилятора необходимо указать системе в необходимых системных переменных пути до исполняемых файлов, заголовочных файлов, файлов библиотек. Для этого используются переменные LDFLAGS, LD\_LIBRARY\_PATH, CPATH, PATH. Реализация экспорта системных переменных представлена в файле export-flags.sh.

## Сборка и установка зависимостей

Типичный сценарий для сборки библиотеки в Unix-подобных системах:

./configure

./make

./make install

Однако не все разработчики придерживаются данного шаблона при разработке своих продуктов, поэтому не удалось выявить универсальную схему для сборки и установки всех зависимостей. К тому же, некоторые библиотеки требуют специальных флагов компиляции, некоторые удобнее собирать утилитой CMake и так далее.

В результате для каждой библиотеки реализован свой сценарий для сборки и установки и представлен в файлах build-\*.sh.

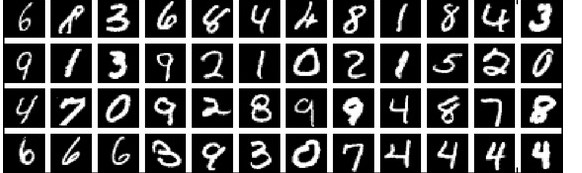
## ***Сборка библиотеки Caffe***

Сборка библиотеки Caffe ничем принципиально не отличается от сборки какой-либо другой библиотеки, за исключением того, что в переменных LD\_LIBRARY\_PATH, CPATH, PATH, LIBS должны быть прописаны корректные значения и пути до собранных зависимостей. Также сборку Caffe отличает то, что необходимо заранее изменить файл конфигурации Makefile.config в зависимости от своей системы или необходимостей. Так, например, имеется возможность выбора реализации интерфейса BLAS, возможность сборки Caffeв CPU-only режиме без CUDA, а также устанавливать нетипичные пути до конкретных библиотек.

Скриптовый сценарий для сборки и установки Caffe представлен в файле build-caffe.sh.

# Тестовые запуски

В составе библиотеки Caffe имеется несколько уже готовых примеров для загрузки тренировочных и тестовых данных и обучения нейронных сетей. Одна из наиболее известных задач классификации – распознавание рукописных цифр из базы данных MNIST [16]. Эта задача была решена в [1] на основе самостоятельно разработанной библиотеки с использованием многослойного персептрона с обратным распространением ошибки и в [2] с помощью библиотеки Tensorflow. В данном разделе будет рассмотрен способ обучения глубокой нейронной сети на базе данных MNIST с использованием библиотеки Caffe.



1. Примеры изображений из базы данных MNIST [18]

В процессе работы с Linux сегментом кластера «Лобачевский» и планировщиком задач SLURM были выявлены некоторые особенности:

* Библиотеки HDF5, LevelDB и LMDB используют механизм блокировки файлов в своем ядре, однако на кластере попытка применить вызов функции lockf() к какому-либо файлу в разделе /home файловой системы приведет к ошибке function not implemented. Потому файлы с данными для обучения необходимо заранее перенести, например, в каталог /tmp/.
* При работе с SLURM можно столкнуться с проблемами линковки библиотек из /usr/lib, так как на вычислительных узлах, возможно, этих библиотек нет. Поэтому был разработан скрипт copy-libs.sh, копирующий недостающие библиотеки в рабочие директории, которые уже заранее были добавлены в LD\_LIBRARY\_PATH.

Конфигурация сети для классификации рукописных цифр описана в [17]. В приложении 1 приведён список команд для загрузки данных, их преобразования к формату mdb и тренировки сети.

Результаты, которые удалось получить после тестирования натренированной сети LeNet:

Test net output #0: accuracy = 0.99

Test net output #1: loss = 0.0295025 (\* 1 = 0.0295025 loss)

Если сравнить полученные результаты и результаты, приведенные в предыдущих работах [1], [2] (accuracy = 0.9428 при использовании многослойного персептрона и accuracy = 0.9079 при использовании методов логистической регрессии), можно сделать очевидный вывод об эффективности и точности решений с использованием глубокого обучения.

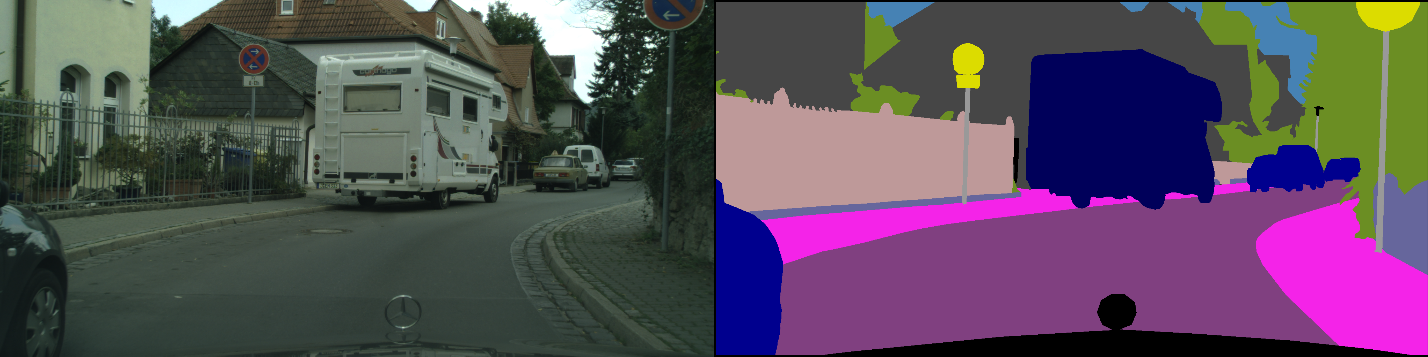
# Процесс работы с библиотекой Caffe

## Набор тренировочных и тестовых данных

В отчете [2] было проведено сравнение нескольких баз изображений для задач компьютерного зрения. Среди них наиболее интересным и подходящим для данной работы оказалась база Cityscapes.

Cityscapes – объемная база [19] изображений дорожных сцен. Все изображения получены с камеры, установленной на лобовом стекле автомобиля в различных городах и условиях съемки.

* Тренировочная выборка: 2975 изображений с точной разметкой и 20000 с грубой.
* Тестовая выборка: 2025 изображений.
* Формат разметки: попиксельный.
* Классы сегментируемых объектов: более 30 классов. Основные группы классов: дорога, люди, транспорт, здания, объекты типа знаков, растительность, небо.



1. Пример семантической сегментации изображения дорожной сцены, взятого из базы Cityscapes [19]

Структура базы тренировочных данных Cityscapes

База Cityscapes содержит не только изображения и разметку к ним, но и видеоматериалы, а также большой набор изображений с грубой разметкой. В данной работе будет рассмотрен только основной набор из 5000 изображений с точной разметкой.

* leftImg8bit – исходные изображения.
  + train – данные, обрабатываемые в фазе тренировки сети.
  + val – данные, обрабатываемые в фазе тестирования, для которые существует разметка.
  + test – данные для проведения тестирования на машинах авторов базы. Разметка недоступна.
* gtFine\_trainvaltest– директория, содержащая данные разметки.
  + train
  + val
  + test

В каждой из директорий train, val и test содержатся поддиректории с названиями городов, в которых располагается некоторое количество изображений.

## Преобразование набора данных к hdf5 формату

Для создания попарного списка с относительными путями до изображений с их разметкой реализован скрипт $CAFFE\_ROOT/cityscapes/get\_data\_list.py (приложение 2).

Наборы данных для задач семантической сегментации представляют собой пары изображения-разметка. Для того чтобы преобразовать исходные данные к удобному представлению для входного слоя в Caffe можно предложить множество вариантов, однако, воспользуемся библиотекой hdf5 для python. Для данного преобразования реализована программа $CAFFE\_ROOT/cityscapes/create\_hdf5.py (приложение 3).

Что сконфигурировать сеть с входными данными в нужном формате создадим входной слой:

layer {

name: "hdf5"

type: "HDF5Data"

top: "data"

top: "label"

hdf5\_data\_param {

source: "cityscapes/train\_h5\_list.txt"

batch\_size: 4

}

include { phase:TRAIN }

}

## Обучение нейронной сети

После создание входного слоя с данными, необходимо указать параметры и конфигурации слоев нейронной сети, после чего начать ее тренировку:

build/tools/caffe train --solver=cityscapes/solver.prototxt

Можно найти уже готовые модели нейронных сетей для пробных исследований. Например, fcn.berkeleyvision [20].

# Заключение

В ходе данной работы выполнены все поставленные задачи, а именно: проведен краткий обзор исследуемой библиотеки Caffe и библиотек, входящих в список ее зависимостей. Для каждой из этих программных библиотек разработан скриптовый сценарий для загрузки архивов исходных кодов, их распаковки, сборки и установки. Все скрипты объединены в единый консольный сценарий так, что вся необходимая работа может быть выполнена путем ввода единственной команды.

После реализации скриптов и успешной установки всех зависимостей и библиотеки Caffe на Linux сегменте кластера «Лобачевский» проведен пробный запуск обучения и тестирования глубокой нейронной сети для базы MNIST.

В процессе работы была изучена структура набора тренировочных данных и разметки Cityscapes для решения задачи семантической сегментации дорожных сцен. Выполнено его преобразование в hdf5 формат, доступные библиотеке Caffe. Проведен процесс обучения сети FCN на этих данных.

В дальнейших работах при прохождении практики планируется разработать улучшения существующих моделей для решения задачи семантической сегментации для набора данных Cityscapes.

Все программные артефакты опубликованы на сайте Github [3].

# Литература

1. Воронцов К. В. Математические методы обучения по прецедентам (теория обучения машин). — 141 с.
2. Хайкин С. Нейронные сети: Полный курс. Пер. с англ. Н. Н. Куссуль, А. Ю. Шелестова. 2-е изд., испр. — М.: Издательский дом Вильямс, 2008, 1103 с.
3. Laurene V. Fausett. Fundamentals of Neural Networks: Architectures, AlgorithmsAndApplications: 1st (first) Edition Paperback — December 9, 1994 — 461 с.
4. Р.В. Шаповалов, Д.П. Ветров, А.А. Осокин, П. Коли Обучение алгоритма семантической сегментации изображений на выборке с разнообразными типами аннотаций.
5. Григорьев Д.С. Обзор алгоритмов семантической сегментации / Д. С. Григорьев, В. Г. Спицын // ХІІІ Всероссийская научно-практическая конференция «Технологии Майкрософт в теории и практике программирования». – 2015. - С. 108-110.
6. Лекция компании Яндекс по нейронным сетям на портале Хабрахабр [<https://habrahabr.ru/company/yandex/blog/307260/>].
7. Cortes C., Vapnik V. Mach Learn (1995) 20: 273 Machine Learning September 1995, Vol. 20, Iss. 3, С. 273–297.
8. Fully Convolutional Models for Semantic Segmentation / Evan Shelhamer\*, Jonathan Long\*, Trevor Darrell / PAMI 2016 /arXiv:1605.06211

# Ссылки

1. Репозиторий, содержащий разработанную программную реализациюи отчетпо учебной практике предыдущего семестра [<https://github.com/Maxim-Doronin/NeuralNetwork>].
2. Ссылка на отчет к производственной практике «Обзор задач компьютерного зрения, возникающих при разработке систем помощи водителю, для решения которых применяются методы глубокого обучения» [<https://github.com/Maxim-Doronin/SemanticSegmentation/blob/master/Курсовая_Доронин.docx>]
3. Репозиторий, содержащий разработанные в процессе учебной практики программы [<https://github.com/Maxim-Doronin/SemanticSegmentation/>]
4. Библиотека Caffe [<http://caffe.berkeleyvision.org>].
5. Библиотека CUDA [<https://developer.nvidia.com/cuda-zone>].
6. Библиотека ATLAS [<http://math-atlas.sourceforge.net/>].
7. Библиотека Boost [<http://www.boost.org/>].
8. Библиотека Protobuf [<https://developers.google.com/protocol-buffers/>].
9. Библиотека Glog [<https://github.com/google/glog>].
10. Библиотека Gflags [<https://gflags.github.io/gflags/>].
11. Библиотека HDF5 [<https://www.hdfgroup.org/>].
12. Библиотека OpenCV [<https://opencv.org/>].
13. Библиотека LMDB [<https://symas.com/lmdb/>].
14. Библиотека LevelDB [<https://github.com/google/leveldb>].
15. Библиотека CMake [<https://cmake.org/>].
16. Ресурс, содержащий базу данных MNIST [[<http://yann.lecun.com/exdb/mnist/>]](http://yann.lecun.com/exdb/mnist).
17. Описание сети LeNetдля обучения на базе данных MNIST [<http://caffe.berkeleyvision.org/gathered/examples/mnist.html>].
18. Ресурс, содержащий изображение данных обучающей выборки [[https://www.researchgate.net/figure/264203784\_fig3\_Samples-from-the-MNIST-database].](https://www.researchgate.net/figure/264203784_fig3_Samples-from-the-MNIST-database%5d.)
19. Статья, описывающая базу Cityscapes [<https://arxiv.org/pdf/1604.01685.pdf>].
20. Fully Convolutional Networks for Semantic Segmentation by Jonathan Long\*, Evan Shelhamer\*, and Trevor Darrell. CVPR 2015 and PAMI 2016. [<http://fcn.berkeleyvision.org/>]

# Приложения

## Обучение сети LeNetна базе данных MNIST

# Download data.

cd $CAFFE\_ROOT

./data/mnist/get\_mnist.sh

Mkdir /tmp/mnist-data

cp ./data/mnist/t10k\* /tmp/mnist-data

cp ./data/mnist/train\* /tmp/mnist-data

EXAMPLE=examples/mnist

DATA=/tmp/mnist-data/

BUILD=build/examples/mnist

BACKEND="lmdb"

rm -rf /tmp/mnist-data/mnist\_train\_${BACKEND}

rm -rf /tmp/mnist-data/mnist\_test\_${BACKEND}

# Convert data to mdb format.

$BUILD/convert\_mnist\_data.bin $DATA/train-images-idx3-ubyte \

$DATA/train-labels-idx1-ubyte /tmp/mnist-data/mnist\_train\_${BACKEND} \

--backend=${BACKEND}

$BUILD/convert\_mnist\_data.bin $DATA/t10k-images-idx3-ubyte \

$DATA/t10k-labels-idx1-ubyte /tmp/mnist-data/mnist\_test\_${BACKEND} \

--backend=${BACKEND}

cp -r /tmp/mnist-data/mnist\* ./examples/mnist/

# Start training.

echo “

#!/usr/bin/envsh

set –e

# Copy data to computer node.

mkdir /tmp/mnist-data

cp -r ./examples/mnist/mnist\_train\_lmdb /tmp/mnist-data

cp -r ./examples/mnist/mnist\_test\_lmdb /tmp/mnist-data

./build/tools/caffe train --solver=examples/mnist/lenet\_solver.prototxt $@

rm -rf /tmp/mnist-data

“ > example/mnist/train\_lenet.sh

sed -i '14s#.\*# source: "/tmp/mnist-data/mnist\_train\_lmdb"#g' ./examples/mnist/lenet\_train\_test.prototxt

sed -i '31s#.\*# source: "/tmp/mnist-data/mnist\_test\_lmdb"#g' ./examples/mnist/lenet\_train\_test.prototxt

# Start training.

srun -N 1 -p gpu./examples/mnist/train\_lenet.sh

## Создание списка изображений из базы данных

import os

data\_path = os.path.join('..', '..', 'data')

for split in ['train', 'val']:

img\_train\_path = os.path.join(data\_path, 'leftImg8bit', split)

img\_train\_list = [os.path.relpath(os.path.join(dp, f), start=data\_path)

fordp, dn, filenames in os.walk(img\_train\_path)

for f in filenames if f.endswith('.png')]

img\_train\_list.sort()

label\_train\_path = os.path.join(data\_path, 'gtFine', split)

label\_train\_list = [os.path.relpath(os.path.join(dp, f), start=data\_path)

fordp, dn, filenames in os.walk(label\_train\_path)

for f in filenames if f.endswith('labelIds.png')]

label\_train\_list.sort()

f = open(os.path.join('eval\_list', split + '.txt'), 'w')

fori in range(0, len(img\_train\_list)):

f.write(img\_train\_list[i] + ' ' + label\_train\_list[i] + '\n')

## Создание hdf5 базы данных

import h5py, os

import caffe

importnumpy as np

import sys

from PIL import Image

IMG\_ROOT=os.path.join('..', '..', 'data')

SIZE = 224 # fixed size to all images

with open( os.path.join('eval\_list', str(sys.argv[1]) + '.txt'), 'r' ) as T :

lines = T.readlines()

X = np.zeros( (len(lines), 3, 512, 256), dtype='f4' )

Y = np.zeros( (len(lines), 1, 512, 256), dtype='f4' )

fori,l in enumerate(lines):

sp = l.split(' ')

img = caffe.io.load\_image( os.path.join(IMG\_ROOT, sp[0]) )

img = caffe.io.resize( img, (512, 256, 3) ) # resize to fixed size

label = Image.open( os.path.join(IMG\_ROOT, sp[1].split()[0]))

label = np.array(label, dtype=np.uint8)

label = caffe.io.resize( label, (512, 256, 1) )

label \*= 255

img \*= 255

transposed\_img = img.transpose((2,0,1))[::-1,:,:] # RGB->BGR

transposed\_label = label.transpose((2,0,1))[::-1,:,:] # RGB->BGR

X[i] = transposed\_img

Y[i] = transposed\_label

ifi % 10 == 0:

printi

ifi> 100:

break

with h5py.File(str(sys.argv[1]) + '.h5','w') as H:

H.create\_dataset( 'data', data=X )

H.create\_dataset( 'label', data=Y )

with open(sys.argv[1] + '\_h5\_list.txt','w') as L:

L.write( '/tmp/cityscapes/' + str(sys.argv[1]) + '.h5' )