Обучение Tesseract 3.03

[Введение 1](#_Toc398710198)

[Дополнительные библиотеки 1](#_Toc398710199)

[Сборка обучающего инструментария 1](#_Toc398710200)

[Необходимые файлы данных 2](#_Toc398710201)

[Требования для входных текстовых файлов 2](#_Toc398710202)

[Что необходимо для начала? 2](#_Toc398710203)

[Процедура обучения 2](#_Toc398710204)

[Генерация обучающих картинок 3](#_Toc398710205)

[Новый автоматизированный метод 3](#_Toc398710206)

[Запуск обучения Tesseract 3](#_Toc398710207)

[Вычисление набора символов 4](#_Toc398710208)

[set\_unicharset\_properties (новое в 3.03) 5](#_Toc398710209)

[font\_properties (новое в 3.01) 5](#_Toc398710210)

[Кластеризация 6](#_Toc398710211)

[Данные словаря (Необязательно) 6](#_Toc398710212)

[Последний файл (unicharambigs) 7](#_Toc398710213)

[Собираем все вместе 9](#_Toc398710214)

[Применение обучения 9](#_Toc398710215)

# Введение

Tesseract 3.0x полностью обучаемый. Данная статья описывает процесс обучения Tesseract, даются некоторые рекомендации по применимости к различным языкам, и чего можно ожидать от результатов.

## Дополнительные библиотеки

Начиная с 3.03, для сборки и инструмента обучения требуются следующие библиотеки:

sudo apt-get install libicu-dev

sudo apt-get install libpango1.0-dev

sudo apt-get install libcairo2-dev

## Сборка обучающего инструментария

Начиная с 3.03, если вы скомпилировали Tesseract из исходных кодов, вам нужно собрать(make) и инсталировать обучающий инструментарий отдельной командой. Необходимы установленные дополнительные библиотеки о котором говорилось выше и выполните следующую команду в папке исходного кода tesseract:

make training

sudo make training-install

## Необходимые файлы данных

Для обучения другого языка, вам необходимо создать файлы данных в подпапке «tessdata», и затем скрипить их вместе в единый файл, используя команду «combine\_tessdata». Именное соглашение – «languagecode.file\_name». Коды языков в релизе соответствуют стандарту «ISO 639-3», но можно использовать любую строку. Файлы, используемые для английского языка (3.00) следующие:

* tessdata/eng.config
* tessdata/eng.unicharset
* tessdata/eng.unicharambigs
* tessdata/eng.inttemp
* tessdata/eng.pffmtable
* tessdata/eng.normproto
* tessdata/eng.punc-dawg
* tessdata/eng.word-dawg
* tessdata/eng.number-dawg
* tessdata/eng.freq-dawg

... и конечный скрипленный файл:

* tessdata/eng.traineddata

и

* tessdata/eng.user-words

… может быть предоставлен дополнительно.

Файл «traineddata» – это конкатенация входных файлов с таблицей содержания, которая содержит смещения файлов. Посмотрите в исходниках «ccutil/tessdatamanager.h» для получения актуального списка принимаемых имен файлов. **ПРИМЕЧАНИЕ** Список файлов в файле «traineddata» отличаются от используемых более ранних версиях, чем 3.00.

## Требования для входных текстовых файлов

Входные текстовые файлы («lang.config», «lang.unicharambigs», «font\_properties», «box files», «wordlists for dictionaries» ...) дожны удовлетворять следующим условиям:

* В ASCII или UTF-8 кодирование без [BOM](http://en.wikipedia.org/wiki/Byte_order_mark/);
* Строки оканчиваются Unix-маркером ('\n');
* Последним символом файла должен быть маркер окончания строки ('\n'). Некоторые текстовые редакторы показывают его как пустую строку в конце файла. Если вы упустли его, то получите сообщение об ошибке «last\_char == '\n': Error: Assert failed...».

## Что необходимо для начала?

Вы должны создать файлы «unicharset», «inttemp», «normproto», «pfftable» используя процедуры, описанные ниже.

# Процедура обучения

Некоторые процедуры неизбежно нужно делать вручную. На сколько возможна автоматизация, насколько она обеспечена. Больше автоматизированных инструментов могут появиться в будущем, но будут требовать дополнительный процесс сборки и инсталяции. Инструменты, упомянаемые ниже – находятся в подпапке обучения.

## Генерация обучающих картинок

Первый шаг – определение всех используемых символов и подготовка файлов с текстом или словами, содержащие набор примеров. Наиболее важные моменты, которые необходимо иметь в виду при создании учебных файлов являются:

* Убедитесь, что есть минимальное количество образцов каждого символа. 10 это хорошо, но 5 пойдет для редких символов;
* Должны быть больше образцов более частых символов - по крайней мере, 20.
* Не делайте ошибки, группируя все небуквенные символы вместе. Делайте текст более реалистичным. Например, «The quick brown fox jumps over the lazy dog. 0123456789 !@#$%^&(),.{}<>/?» ужасный. Намного лучше «The (quick) brown {fox} jumps! over the $3,456.78<lazy> #90 dog & duck/goose, as 12.5% of E-mail from aspammer@website.com is spam?» Это дает гораздо больше шансов получить разумные исходные показатели для специальных символов.

## Новый автоматизированный метод

Подготовьте текстовый файл в utf-8 (training\_text.txt) содержащий ваш обучающий текст в соответствии с приведенной выше требованиям. Скачайте «truetype/opentype» файлы шрифтов, для которых собираетесь обучить распознование. Выполните следующую команду для каждого шрифта для создания пары соответствия файл TIF / BOX.

training/text2image --text=training\_text.txt --outputbase=[lang].[fontname].exp0 --font='Font Name' --fonts\_dir=/path/to/your/fonts

Заметьте, что аргумент «--font» может содержать пробелы, но тем сымым должен быть заключены в апострофы. Т.е.:

training/text2image --text=training\_text.txt --outputbase=eng.TimesNewRomanBold.exp0 --font='Times New Roman Bold' --fonts\_dir=/usr/share/fonts

Есть много других аргументов доступных для «text2image». Смотрите training/text2image.cpp для дополнительной информации.

# Запуск обучения Tesseract

Для каждого обучающей пары картинка-box-файл выполните:

tesseract [lang].[fontname].exp[num].tif [lang].[fontname].exp[num] box.train

… или

tesseract [lang].[fontname].exp[num].tif [lang].[fontname].exp[num] box.train.stderr

**ПРИМЕЧАНИЕ** хотя tesseract на этом этапе требует присутствие языковых данных, данные язык не используется, таким образом будет делаться для английского языка, какой бы язык вы ни обучали.

Первая команда отправляет все ошибки в файл «tesseract.log» (для всех платформ), как это делалось на windows версии 2.03 и ниже. С параметром.stderr, все ошибки будут отправляться на stderr, для всех платформ, как это делалось на не-windows версии 2.03 и ниже.

Заметьте, что имя box-файла должна быть tif, включая путь, или Tesseract не сможет его найти. Результатом этого шага является «fontfile.tr», который содержит характеристики каждого символа из тренировочной картинки. Также создается файл «[lang].[fontname].exp[num].txt» с простым символом «конец строки» и без текста.

**ВАЖНО** Проверьте на ошибки в выходном файле из apply\_box. Если там имеется FATALITIES, то нет никакого смысла продолжения процесса обучения, пока вы не исправить box-файл. Новая конфигурационный файл box.train.stderr позволяет легко выбрать путь к выводу. FATALITY обычно указывает, что этот шаг потерпел неудачу при поиске одного символа указанного в box-файле. Либо координаты неверны, или что-то неладно с изображением символа. Если нет работоспособного образца символа, оно не может быть распознано, и позже сгенерированный inttemp файл не будет соответствовать unicharset файлу и Tesseract прервется.

Также, может возникнуть другая ошика **которая также критична и требует внимания** – ошибка "Box file format error on line n". Если предшествует некорректный код utf-8 и требует исправления. Ошибка "utf-8 string too long..." указывает, что что вы превысили лимит 24 байт на описание символа Если вам нужно описание больше, чем 24 байта, опишите ее в файле.

Нет необходимости редакрировать файл [lang].[fontname].exp[num].tr. Имя шрифта внутри него не должно быть установлено. Для любопытных, вот некоторая информация о формате:

Every character in the box file has a corresponding set of entries in  
the .tr file (in order) like this   
UnknownFont <utf8 code(s)> 2  
mf <number of features>  
x y length dir 0 0  
... (there are a set of these determined by <number of features>  
above)  
cn 1  
ypos length x2ndmoment y2ndmoment  
  
The mf features are polygon segments of the outline normalized to the  
1st and 2nd moments.  
x= x position [-0.5.0.5]  
y = y position [-0.25, 0.75]  
length is the length of the polygon segment [0,1.0]  
dir is the direction of the segment [0,1.0]  
  
The cn feature is to correct for the moment normalization to  
distinguish position and size (eg c vs C and , vs ')

# Вычисление набора символов

Tesseract необходимо знать набор возможных выходных символов. Для генерации файла unicharset, используйте программу unicharset\_extractorна box-файлы, которые сгенерированы выше:

unicharset\_extractor lang.fontname.exp0.box lang.fontname.exp1.box ...

Tesseract должен иметь доступ к свойствам символов isalpha, isdigit, isupper, islower, ispunctuation. Эти данные должны быть в файле «unicharset». Каждая строка файла соответствует одному символу. Символ в UTF-8 следует шестнадцатеричное число, представляющее собой бинарную маску, которая кодирует свойства. Каждый бит соответствует свойству. Если бит установлен 1, это значит, что свойсто истинно. Порядок битов:isalpha, islower, isupper, isdigit, ispunctuation.

Пример:

* ';' символ пунктуации. В бинарном виде 10000 (10 в шестнадцатиричном).
* 'b' буква алфавита и нижний регистр. В бинарном - 00011 (3 в шестнадцатиричном).
* 'W' заглавный символ алфавита. В бинарном 00101 (5 в шестнадцатиричном).
* '7' – только число. 01000 (8 в шестнадцатиричном).
* '=' не алфавит, не цифра не пунктуация. 00000 (0 в шестнадцатиричном).

; 10 Common 46

b 3 Latin 59

W 5 Latin 40

7 8 Common 66

= 0 Common 93

Если ваша система поддерживает функции wctype, эти значения проставяться автоматически программой unicharset\_extractor **и нет необходимости редактировать файл unicharset.** На некоторых системах, такие как Windows 95, треубется редактирование файл unicharset вручную.

**ПРИМЕЧАНИЕ** Файл «unicharset» должен быть перегенерен каждый раз после генерации файлов «inttemp», «normproto» и «pffmtable» (т.е. они должны все пересозданы, когда box-файл изменен), чтобы они должны быть синхронизированы. Это облегчено, чем в предыдущих версиях, запустив unicharset\_extractor до mftraining и cntraining, и передавая unicharset в mftraining.

Последние два столбца представляют тип скрипта (Latin, Common, Greek, Cyrillic, Han, null) и идентификатор кода символа данного языка.

## set\_unicharset\_properties (новое в 3.03)

Новый инструмент и набор файлов данных в 3.03 позволяют добавление дополнительных свойств в unicharset, в основном те размеры шрифтов.

training/set\_unicharset\_properties -U input\_unicharset -O output\_unicharset --script\_dir=training/langdata

## font\_properties (новое в 3.01)

Появилось новое требование для обучения в 3.01 – файл «font\_properties». Цель этого файла – проедоставление информации о стиле шрифта, которые появляются на выходе, когда шрифт распознан. Файл «font\_properties» - текстовый файл указываемый параметром «-F filename» для «mftraining».

Каждая строка файла «font\_properties» имеет следующий формат:

<fontname> <italic> <bold> <fixed> <serif> <fraktur>

где <fontname> строковое название шрифта (пробелы не допускаются!), и <italic>, <bold>, <fixed>, <serif> and <fraktur> - могут принимать значения 0 или 1, которые указываю имеет ли данный шрифт это свойсто.

Когда выполняется «mftraining», каждое название файлов «.tr» должны соответствовать must match an записи в файле «font\_properties», или «mftraining» прервется.

**Example:** файл «font\_properties»:

timesitalic 1 0 0 1 0

shapeclustering -F font\_properties -U unicharset eng.timesitalic.exp0.tr  
mftraining -F font\_properties -U unicharset -O eng.unicharset eng.timesitalic.exp0.tr

**Примечание** в 3.03, по умолчанию есть файл «font\_properties», который охватывает 3000 шрифтов (не обязательно точно) «intraining» / «langdata» / «font\_properties».

## Кластеризация

Когда извлечены все особенности символов для всех обучаемых страниц, вам необходимо кластеризовать их для создания прототипов (prototypes). Особенности формы символ можно сгруппировать с помощью программ «shapeclustering» (доступен с версии 3,02), «mftraining» и «cntraining» программы:

shapeclustering -F font\_properties -U unicharset lang.fontname.exp0.tr lang.fontname.exp1.tr ...

«shapeclustering» создает главную таблицу формы по форме кластеров и записывает его в файл «shapetable».

**Примечание**: Если вы получили сообщение об ошибке «"index >= 0 && index < size\_used\_:Error:Assert failed in genericvector.h, line 512"» - добавте файл «shapetable» файл в файла данных вашего языка.

mftraining -F font\_properties -U unicharset -O lang.unicharset lang.fontname.exp0.tr lang.fontname.exp1.tr ...

Файл в параметре «-U» - unicharset, который генериться командой «unicharset\_extractor», и «lang.unicharset» является выход unicharset что будет дан для «combine\_tessdata». «Mftraining» выводит другие файлы: «inttemp» (the shape prototypes) «pffmtable» (the number of expected features for each character).

cntraining lang.fontname.exp0.tr lang.fontname.exp1.tr ...

Это выводит файл «normproto» (the character normalization sensitivity prototypes).

## Данные словаря (Необязательно)

Tesseract uses up to 8 dictionary files for each language. These are all optional, and help Tesseract to decide the likelihood of different possible character combinations.

Seven of the files are coded as a Directed Acyclic Word Graph (DAWG), and the other is a plain UTF-8 text file:

To make the DAWG dictionary files, you first need a wordlist for your language. You may find an appropriate dictionary file to use as the basis for a wordlist from the spellcheckers (e. g. [ispell](http://ficus-www.cs.ucla.edu/geoff/ispell-dictionaries.html), [aspell](http://aspell.net/) or [hunspell](http://hunspell.sourceforge.net/)) - be careful about the license. The wordlist is formatted as a UTF-8 text file with one word per line. Split the wordlist into needed sets e.g.: the frequent words, and the rest of the words, and then use wordlist2dawg to make the DAWG files:

|  |  |  |
| --- | --- | --- |
| **Name** | **Type** | **Description** |
| word-dawg | dawg | A dawg made from dictionary words from the language. |
| freq-dawg | dawg | A dawg made from the most frequent words which would have gone into word-dawg. |
| punc-dawg | dawg | A dawg made from punctuation patterns found around words. The *"word"* part is replaced by a single space. |
| number-dawg | dawg | A dawg made from tokens which originally contained digits. Each digit is replaced by a space character. |
| fixed-length-dawgs | dawg | Several dawgs of different fixed lengths —— useful for languages like Chinese. |
| bigram-dawg | dawg | A dawg of word bigrams where the words are separated by a space and each digit is replaced by a *?*. |
| unambig-dawg | dawg | TODO: Describe. |
| user-words | text | A list of extra words to add to the dictionary. Usually left empty to be added by users if they require it; see [tesseract(1)](http://tesseract-ocr.googlecode.com/svn/trunk/doc/tesseract.1.html" \l "_config_files_and_augmenting_with_user_data). |

wordlist2dawg frequent\_words\_list lang.freq-dawg lang.unicharset  
wordlist2dawg words\_list lang.word-dawg lang.unicharset

For right-to-left languages (RTL) use option *"-r 1"*. Other options can be found in [wordlist2dawg Manual Page](http://tesseract-ocr.googlecode.com/svn-history/trunk/doc/wordlist2dawg.1.html)

**NOTE:** If a dictionary file is included in the combined traineddata, it must contain at least one entry. Dictionary files that would otherwise be empty are not required for the combine\_tessdata step. Words with unusual spellings should be added to the dictionary files. Unusual spellings can include mixtures of alphabetical characters with punctuation or numeric characters. (E.g. i18n, l10n, google.com, news.bbc.co.uk, io9.com, utf8, ucs2)

If you need example files for dictionary wordlists, uncombine (with [combine\_tessdata](http://tesseract-ocr.googlecode.com/svn-history/r800/trunk/doc/combine_tessdata.1.html)) existing language data file (e.g. eng.traineddata) and then extract wordlist with [dawg2wordlist](http://tesseract-ocr.googlecode.com/svn-history/trunk/doc/dawg2wordlist.1.html)

# Последний файл (unicharambigs)

Последний файл, который используется Tesseract-ом, называется unicharambigs. Он описывает возможные двусмысленности между символами или набором символов, и формируется вручную. Для понимания его формата, рассмотрим пример:

v1

2 ' ' 1 " 1

1 m 2 r n 0

3 i i I 1 m 0

Первая строка – номер версии. Остальные строки разделены поля ТАБ-ом, в следующем формате: <количество символов для источника>:

**<tab>**

<characters for match source>

**<tab>**

<number of characters for match target>

**<tab>**

<characters for match target>

**<tab>**

<type indicator>

Указатель типа может иметь следующие значения:

|  |  |  |
| --- | --- | --- |
| **Значение** | **Тип** | **Описание** |
| 0 | A non-mandatory substitution. This informs tesseract to consider the ambiguity as a hint to the segmentation search that it should continue working if replacement of 'source' with 'target' creates a dictionary word from a non-dictionary word. Dictionary words that can be turned to another dictionary word via the ambiguity will not be used to train the adaptive classifier. |  |
| 1 | A mandatory substitution. This informs tesseract to always replace the matched 'source' with the 'target' strings. |  |

|  |  |
| --- | --- |
| **Example line** | **Explanation** |
| 2 ' ' 1 " 1 | A double quote (") should be substituted **whenever** 2 consecutive single quotes (') are seen. |
| 1 m 2 r n 0 | The characters 'rn' may sometimes be recognized incorrectly as 'm'. |
| 3 i i i 1 m 0 | The character 'm' may sometimes be recognized incorrectly as the sequence 'iii'. |

Each separate character must be included in the unicharset. That is, all of the characters used must be part of the language that is being trained.

The rules are not bidirectional, so if you want 'rn' to be considered when 'm' is detected and vise versa you need a rule for each.

Version 3.03 and on supports a new, simpler format for the unicharambigs file:

v2  
'' " 1  
m rn 0  
iii m 0

In this format, the "error" and "correction" are simple utf-8 strings separated by a space, and, after another space, the same type specifier as v1 (0 for optional and 1 for mandatory substitution). Note the downside of this simpler format is that Tesseract has to encode the utf-8 strings into the components of the unicharset. In complex scripts, this encoding may be ambiguous. In this case, the encoding is chosen such as to use the least utf-8 characters for each component, ie the shortest unicharset components will make up the encoding.

Like most other files used in training, the 'unicharambigs' file must be encoded as UTF8, and must end with a newline character. The unicharambigs format is also described in the [unicharambigs(5) man page](https://tesseract-ocr.googlecode.com/svn-history/r683/trunk/doc/unicharambigs.5.html).

The unicharambigs file may also be non-existent.

# Собираем все вместе

Вам нужно собрать вместе все (shapetable, normproto, inttemp, pffmtable) файлы и переименовать их с «alang prefix», где язык является 3-буквенный код для вашего языка взяты из http://en.wikipedia.org/wiki/List\_of\_ISO\_639-2\_codes а затем запустить combine\_tessdata следующим образом:

combine\_tessdata lang.

**ПРИМЕЧАНИЯ: Не забывайте точку в конце!**

Результат - файл «lang.traineddata» положить в папку «tessdata». Tesseract может расспознавать текст вашего языка следующим образом:

tesseract image.tif output -l lang

Дополнительные параметры «combine\_tessdata» можно найти на странице «Руководство» или в комментариях исходного кода.

# Применение обучения

1) Формирование картинок и box-файлов для обучения

../tesseract-source/tesseract-ocr/training/text2image --text=kazText.txt --outputbase=kaz.Arial.exp0 --font='Arial' --fonts\_dir=/usr/share/fonts/truetype/msttcorefonts/

-- RESULT:

kaz.Arial.exp0.box

kaz.Arial.exp0.tif

2)

tesseract kaz.Arial.exp0.tif kaz.Arial.exp0 box.train

-- RESULT:

kaz.Arial.exp0.tr

kaz.Arial.exp0.txt

3)

unicharset\_extractor kaz.Arial.exp0.box

-- RESULT:

unicharset

4)

../tesseract-source/tesseract-ocr/training/set\_unicharset\_properties -U input\_unicharset -O output\_unicharset --script\_dir=../tesseract-source/tesseract-ocr/training/langdata

5)

shapeclustering -F kaz.font\_properties -U unicharset kaz.Arial.exp0.tr

-- RESULT:

shapetable

6)

mftraining -F kaz.font\_properties -U unicharset -O kaz.unicharset kaz.Arial.exp0.tr

-- RESULT:

inttemp

kaz.unicharset

shapetable

pffmtable

7)

cntraining kaz.Arial.exp0.tr

normproto

8)

combine\_tessdata kaz.

Путь к tessdata:

/usr/local/share/tessdata

Установка Imagemagick

Обновляем информацию о доступных пакетах

apt-get update

Производим установку

apt-get install imagemagick

Проверяем установленную версию

convert -version