(https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract-3.03%E2%80%933.05/)

Training Tesseract 3.03–3.05

Ryan Conway edited this page on 9 Jul 2018 · 4 revisions

How to use the tools provided to train Tesseract 3.03–3.05 for a new language. Tesseract 3.03から3.05を新しい言語用にトレーニングするために提供されているツールの使用方法。

Important note: Before you invest time and efforts on training Tesseract, it is highly recommended to read the ImproveQuality page.
Tesseract 3.04 and 3.05 provide a script for an easy way to execute the various phases of training Tesseract. More information on using it can be found on the tesstrain.sh page.
重要な注意：Tesseractのトレーニングに時間と努力を注ぐ前に、ImproveQualityページを読むことを強くお勧めします。
Tesseract 3.04と3.05は、Tesseractをトレーニングするさまざまな段階を実行するための簡単な方法のためのスクリプトを提供します。 それを使用することに関するより多くの情報はtesstrain.shページで見つけることができます。

Questions about the training process トレーニングプロセスに関する質問

If you had some problems during the training process and you need help, use tesseract-ocr mailing-list to ask your question(s).
PLEASE DO NOT report your problems and ask questions about training as issues!
トレーニングの過程で問題があり、助けが必要な場合は、tesseract-ocr mailing-listを使って質問をしてください。問題を報告したり、問題としてトレーニングについて質問したりしないでください。

Introduction 前書き

Tesseract 3.0x is fully trainable. This page describes the training process, provides some guidelines on applicability to various languages, and what to expect from the results.
Please check the list of languages for which traineddata is already available as of release 3.04 before embarking on training.
3rd Party training tools are also available for training.
Tesseract 3.0xは完全にトレーニング可能です。このページでは、トレーニングプロセスについて説明し、さまざまな言語への適用性に関するガイドライン、および結果から何を期待するかについて説明します。
トレーニングを開始する前に、リリース3.04以降にすでにトレーニング済みデータが使用可能な言語のリストを確認してください。
サードパーティのトレーニングツールもトレーニングに利用できます。

Background and Limitations背景と制限

Tesseract was originally designed to recognize English text only. Efforts have been made to modify the engine and its training system to make them able to deal with other languages and UTF-8 characters. Tesseract 3.0 can handle any Unicode characters (coded with UTF-8), but there are limits as to the range of languages that it will be successful with, so please take this section into account before building up your hopes that it will work well on your particular language!
Tesseractはもともと英語のテキストだけを認識するように設計されていました。エンジンとそのトレーニングシステムを変更して、他の言語とUTF-8文字を処理できるようにする努力がなされています。
Tesseract 3.0は（UTF-8でコード化された）どんなUnicode文字も扱うことができますが、それがうまくいく言語の範囲に関して制限があるので、それがうまくいくことを期待する前にこのセクションを考慮に入れてくださいあなたの特定の言語！

Tesseract 3.01 added top-to-bottom languages, and Tesseract 3.02 added Hebrew (right-to-left).
Tesseract 3.01は上から下への言語を追加し、Tesseract 3.02はヘブライ語（右から左へ）を追加しました。

Tesseract currently handles scripts like Arabic and Hindi with an auxiliary engine called cube (included in Tesseract version 3.01 and up). **Don't try to train Tesseract versions earlier than 4.0 for Arabic (same for Persian, Urdu, etc.). It's hopeless.** For 4.0 only train with the LSTM method.
Tesseractは現在、cubeと呼ばれる補助エンジンを備えたアラビア語やヒンディー語などのスクリプトを処理します（Tesseractバージョン3.01以降に含まれています）。 Tesseractのバージョンが4.0より前のアラビア語を訓練しないでください（ペルシャ語、ウルドゥー語なども同じ）。それは絶望的です。 4.0の場合は、LSTM方式でのみトレーニングしてください。

Traineddata for additional languages has been provided by Google for the 3.04 release.
3.04リリースでは、追加言語用のトレーニングデータがGoogleから提供されています。

Tesseract is slower with large character set languages (like Chinese), but it seems to work OK.
Tesseractは（中国語のような）大きな文字セット言語では遅くなりますが、うまくいくようです。

Tesseract needs to know about different shapes of the same character by having different fonts separated explicitly. The number of fonts is limited to 64 fonts. Note that runtime is heavily dependent on the number of fonts provided, and training more than 32 will result in a significant slow-down.
Tesseractは、異なるフォントを明示的に分離することによって、同じ文字の異なる形状について知る必要があります。フォント数は64フォントに制限されています。実行時間は提供されるフォントの数に大きく依存します。32以上のトレーニングは大幅に遅くなります。

Additional Libraries required　必要な追加ライブラリ

Beginning with 3.03, additional libraries are required to build the training tools.
3.03以降、トレーニングツールを構築するために追加のライブラリが必要になりました。

```
sudo apt-get install libicu-dev
sudo apt-get install libpango1.0-dev
sudo apt-get install libcairo2-dev
```

Building the training tools　トレーニングツールを構築する

Beginning with 3.03, if you're compiling Tesseract from source you need to make and install the training tools with separate make commands. Once the above additional libraries have been installed, run the following from the tesseract source directory:
3.03から、Tesseractをソースからコンパイルしている場合は、別々のmakeコマンドでトレーニングツールを作成してインストールする必要があります。上記の追加ライブラリがインストールされたら、tesseractソースディレクトリから以下を実行します。

```
make training
sudo make training-install
```

Data files required　必要なデータファイル

To train for another language, you have to create some data files in the tessdata subdirectory, and then crunch these together into a single file, using combine_tessdata. The naming convention is languagecode.file_name Language codes for released files follow the ISO 639-3 standard, but any string can be used. The files used for English (3.0x) are:
別の言語にトレーニングするには、tessdataサブディレクトリにデータファイルをいくつか作成してから、combine_tessdataを使用してこれらをまとめて1つのファイルにする必要があります。命名規則はlanguagecode.file_nameリリースされたファイルの言語コードはISO 639-3規格に準拠していますが、任意の文字列を使用できます。英語（3.0x）に使用されるファイルは次のとおりです。

- 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

... and the final crunched file is: ...そして最後のクランチファイルは次のとおりです。

- tessdata/eng.traineddata

and そして

- tessdata/eng.user-words may still be provided separately. は、引き続き別に提供される場合があります。

The traineddata file is simply a concatenation of the input files, with a table of contents that contains the offsets of the known file types. See ccutil/tessdatamanager.h in the source code for a list of the currently accepted filenames.
学習済みデータファイルは、既知のファイルタイプのオフセットを含む目次を含む入力ファイルの単なる連結です。現在受け入れられているファイル名のリストについては、ソースコードのccutil / tessdatamanager.hを参照してください。

Requirements for text input files テキスト入力ファイルの要件

Text input files (lang.config, lang.unicharambigs, font_properties, box files, wordlists for dictionaries...) need to meet these criteria:
テキスト入力ファイル（lang.config、lang.unicharambigs、font_properties、ボックスファイル、辞書の単語リスト...）は、次の基準を満たす必要があります。

- ASCII or UTF-8 encoding without BOM
- BOMなしのASCIIまたはUTF-8エンコーディング

- Unix end-of-line marker ('¥n')
- Unixの行末マーカー ('¥n')

- The last character must be an end of line marker ('¥n'). Some text editors will show this as an empty line at the end of file. If you omit this you will get an error message containing last_char == '¥n':Error:Assert failed....
- 最後の文字は行末マーカー（ '¥ n'）でなければなりません。 テキストエディタの中には、これをファイルの最後に空行として表示するものがあります。 これを省略すると、last_char == '¥ n'を含むエラーメッセージが表示されます。エラー：アサートに失敗しました....

How little can you get away with? あなたはどのくらい逃げることができますか？

You **must** create unicharset, inttemp, normproto, pffmtable using the procedure described below. If you are only trying to recognize a limited range of fonts (like a single font for instance), then a single training page might be enough. The other files do not need to be provided, but will most likely improve accuracy, depending on your application.
下記の手順で、unicharset、inttemp、normproto、pffmtableを作成する**必要があります**。 限られた範囲のフォント（たとえば単一のフォントなど）のみを認識しようとしている場合は、単一のトレーニングページで十分かもしれません。 他のファイルを提供する必要はありませんが、アプリケーションによっては正確性が向上する可能性があります。

Training Procedure トレーニング手順

Some of the procedure is inevitably manual. As much automated help as possible is provided. The tools referenced below are all built in the training subdirectory.
手順のいくつかは必然的に手動です。できるだけ多くの自動ヘルプが提供されています。下記のツールはすべてtrainingサブディレクトリに構築されています。

You need to run all commands in the same folder where your input files are located.
入力ファイルが置かれているのと同じフォルダーですべてのコマンドを実行する必要があります。

Generate Training Images and Box Files
トレーニング画像とボックスファイルを生成する

Prepare a text file テキストファイルを準備する

The first step is to determine the full character set to be used, and prepare a text or word processor file containing a set of examples. The most important points to bear in mind when creating a training file are:
最初のステップは、使用する全文字セットを決定し、一連の例を含むテキストまたはワードプロセッサファイルを準備することです。トレーニングファイルを作成するときに留意する必要がある最も重要な点は次のとおりです。

Make sure there are a minimum number of samples of each character. 10 is good, but 5 is OK for rare characters.
各文字のサンプル数が最小であることを確認してください。 10が良いですが、5はまれな文字のためにOKです。

There should be more samples of the more frequent characters - at least 20.
より頻繁な文字のより多くのサンプルがあるはずです - 少なくとも20。

Don't make the mistake of grouping all the non-letters together. Make the text more realistic.
すべての文字以外をまとめて間違えないでください。テキストをよりリアルにします。

For example: 例えば:

```
The quick brown fox jumps over the lazy dog.
0123456789 !@#$%^&(),.{}&lt;&gt;/?
```

is terrible! ひどいです！

Much better is: はるかに良いです:

```
The (quick) brown {fox} jumps! over the $3,456.78 &lt;lazy&gt; #90 dog &
duck/goose, as 12.5% of E-mail from aspammer@website.com is spam?
```

This gives the text-line finding code a much better chance of getting sensible baseline metrics for the special characters.
これにより、テキスト行検出コードは、特殊文字の実用的なベースラインメトリックを取得する機会が大幅に増えます。

Automated method 自動化された方法

New in 3.03 3.03の新機能

Prepare a UTF-8 text file (training_text.txt) containing your training text according to the above specification. Obtain truetype/opentype font files for the fonts that you wish to recognize. Run the following command for each font in turn to create a matching tif/box file pair.
上記の指定に従って、トレーニングテキストを含むUTF-8テキストファイル（training_text.txt）を用意してください。認識したいフォントのTrueType / OpenTypeフォントファイルを入手します。各フォントに対して次のコマンドを順番に実行して、一致するtif / boxファイルのペアを作成します。

```
training/text2image --text=training_text.txt
--outputbase=[lang].[fontname].exp0 --font='Font Name'
--fonts_dir=/path/to/your/fonts
```

Note that the argument to --font may contain spaces, and thus must be quoted. Eg:
--fontの引数にはスペースを含めることができるため、引用符で囲む必要があります。例えば:

```
training/text2image --text=training_text.txt
--outputbase=eng.TimesNewRomanBold.exp0 --font='Times New Roman Bold'
--fonts_dir=/usr/share/fonts
```

To list all fonts in your system which can render the training text, run:
トレーニングテキストを表示できるシステム内のすべてのフォントを一覧表示するには、次のコマンドを実行します。

```
training/text2image --text=training_text.txt --outputbase=eng
--fonts_dir=/usr/share/fonts  --find_fonts --min_coverage=1.0
--render_per_font=false
```

In this example, the training_text.txt file contains text written in English. A 'eng.fontlist.txt' file will be created.
この例では、training_text.txtファイルには英語で書かれたテキストが含まれています。
'eng.fontlist.txt'ファイルが作成されます。

There are a lot of other command-line arguments available to text2image. Run text2image --help to get more information.
text2imageには他にもたくさんのコマンドライン引数があります。詳細については、text2image --helpを実行してください。

If you used text2image, you can move to Run Tesseract for Training step.
text2imageを使用した場合は、「トレーニングのためにTesseractを実行する」ステップに進むことができます。

Old Manual method 旧マニュアル方式

- The training data should be grouped by font. Ideally, all samples of a single font should go in a single tiff file, but this may be multi-page tiff (if you have libtiff or leptonica installed), so the total training data in a single font may be many pages and many 10s of thousands of characters, allowing training for large-character-set languages.
- トレーニングデータはフォントごとにまとめてください。理想的には、単一フォントのすべてのサンプルは単一TIFFファイルに入れるべきですが、これは複数ページTIFF（libtiffまたはleptonicaがインストールされている場合）なので、単一フォントの合計トレーニングデータは多数のページと数10大規模な文字セットの言語のトレーニングが可能

- There is no need to train with multiple sizes. 10 point will do. (An exception to this is very small text. If you want to recognize text with an x-height smaller than about 15 pixels, you should either train it specifically or scale your images before trying to recognize them.)
- 複数のサイズでトレーニングする必要はありません。 10点になります。（これに対する例外は非常に小さいテキストです。もしあなたが約15ピクセルよりも小さいxの高さを持つテキストを認識したいのなら、それらを認識することを試みる前にあなたはそれを特別に訓練するかあなたのイメージを拡大縮小するべきです

- **DO NOT MIX FONTS IN AN IMAGE FILE** (In a single .tr file to be precise.) This will cause features to be dropped at clustering, which leads to recognition errors.
- **単一のイメージファイルに複数のフォントを混ぜないでください**（正確には1つの.trファイルにまとめます）。これにより、クラスタリング時にフィーチャが削除され、認識エラーが発生します。

- The example boxtiff files on the downloads page will help if you are not sure how to format your training data.
- ダウンロードページのサンプルのboxtiffファイルは、トレーニングデータのフォーマット方法がわからない場合に役立ちます。

Next print and scan (or use some electronic rendering method) to create an image of your training page. Up to 64 training files can be used (of multiple pages). It is best to create a mix of fonts and styles (but in separate files), including italic and bold.
次に印刷してスキャンし（または何らかの電子レンダリング方法を使用して）、トレーニングページの画像を作成します。最大64個のトレーニングファイル（複数ページ）を使用できます。イタリック体とボールド体を含め、フォントとスタイルを組み合わせて（ただし別々のファイルに）作成するのが最善です。

You will also need to save your training text as a UTF-8 text file for use in the next step where you have to insert the codes into another file.
また、トレーニングテキストをUTF-8テキストファイルとして保存して、次のステップで使用するコードを別のファイルに挿入する必要があります。

Clarification for large amounts of training data The 64 images limit is for the number of **FONTS.** Each font should be put in a single multi-page tiff and the box file can be modified to specify the page number for each character after the coordinates. Thus an arbitrarily large amount of training data may be created for any given font, allowing training for large character-set languages. An alternative to multi-page tiffs is to create many single-page tiffs for a single font, and then you must cat together the tr files for each font into several single-font tr files. In any case, the input tr files to mftraining must each contain a single font.
大量のトレーニングデータの説明 64イメージの制限はFONTSの数に対するものです。各フォントは単一の複数ページのTIFFに入れられるべきであり、ボックスファイルは座標の後の各文字のページ番号を指定するように修正することができます。したがって、任意のフォントに対して任意の量のトレーニングデータを作成でき、大きな文字セット言語のトレーニングが可能になります。複数ページのTIFFに代わる方法は、1つのフォントに対して複数の単一ページのTIFFを作成することです。その後、各フォントのtrファイルをいくつかの単一フォントのtrファイルにまとめる必要があります。いずれにせよ、mftrainingへの入力trファイルはそれぞれ単一のフォントを含まなければなりません。

Make Box Files ボックスファイルを作る

See the separate Make Box Files page.
別の「ボックスファイルの作成」ページを参照してください。
(https://github.com/tesseract-ocr/tesseract/wiki/Make-Box-Files)

Run Tesseract for Training
トレーニングのためにTesseractを実行する

For each of your training image, boxfile pairs, run Tesseract in training mode:
トレーニング画像、ボックスファイルのペアごとに、Tesseractをトレーニングモードで実行します。

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

or または

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

NOTE that although tesseract requires language data to be present for this step, the language data is not used, so English will do, whatever language you are training.
tesseractはこのステップのために言語データが存在することを要求しますが、言語データは使用されないので、あなたが訓練しているどんな言語でも、英語はそうするでしょう。

The first form sends all the errors to a file named tesseract.log. The second form sends all errors to stderr.
最初の形式はすべてのエラーをtesseract.logという名前のファイルに送ります。 2番目の形式はすべてのエラーをstderrに送ります。

Note that the box filename must match the tif filename, including the path, or Tesseract won't find it. The output of this step is fontfile.tr which contains the features of each character of the training page. [lang].[fontname].exp[num].txt will also be written with a single newline and no text.
ボックスfilenameはパスを含むtifファイル名と一致しなければならないことに注意してください。そうしないとTesseractはそれを見つけられません。このステップの出力は、トレーニングページの各文字の機能を含むfontfile.trです。 [lang]。[fontname] .exp [num] .txtも1行の改行でテキストは表示されません。

Important Check for errors in the output from apply_box. If there are FATALITIES reported, then there is no point continuing with the training process until you fix the box file. The new box.train.stderr config file makes is easier to choose the location of the output. A FATALITY usually indicates that this step failed to find any training samples of one of the characters listed in your box file. Either the coordinates are wrong, or there is something wrong with the image of the character concerned. If there is no workable sample of a character, it can't be recognized, and the generated inttemp file won't match the unicharset file later and Tesseract will abort.
重要 apply_boxからの出力にエラーがないか確認してください。死亡が報告されている場合は、ボックスファイルを修正するまでトレーニングプロセスを続行する意味はありません。新しいbox.train.stderr設定ファイルにより、出力の場所を選択しやすくなりました。 FATALITYは通常、このステップであなたのボックスファイルにリストされているキャラクターの1つのトレーニングサンプルが見つからなかったことを示します。座標が間違っているか、関係するキャラクターの画像に何か問題があります。実行可能な文字のサンプルがなければ、それは認識されず、生成されたinttempファイルは後でunicharsetファイルと一致せず、Tesseractは中止されます。

Another error that can occur **that is also fatal and needs attention** is an error about "Box file format error on line n". If preceded by "Bad utf-8 char..." then the UTF-8 codes are incorrect and need to be fixed. The error "utf-8 string too long..." indicates that you have exceeded the 24 byte limit on a character description. If you need a description longer than 24 bytes, please file an issue.
また致命的であり、注意が必要な、発生する可能性があるもう1つのエラーは、「行nのボックスファイル形式エラー」に関するエラーです。 "Bad utf-8 char ..."が前に付いている場合、UTF-8コードは正しくないため修正する必要があります。エラー "utf-8 string too long ..."は、文字の説明で24バイトの制限を超えたことを示します。 24バイトを超える説明が必要な場合は、問題を報告してください。
There is no need to edit the content of the [lang].[fontname].exp[num].tr file. The font name inside it need not be set.
[lang].[fontname].exp[num].trファイルの内容を編集する必要はありません。その中のフォント名を設定する必要はありません。

For the curious, here is some information on the format.
好奇心旺盛な人のために、ここでフォーマットに関するいくつかの情報があります。
(https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract-3.03%E2%80%933.05#the-tr-file-format)

Generate the unicharset file unicharsetファイルを生成する

Tesseract's unicharset file contains information on each symbol (unichar) the Tesseract OCR engine is trained to recognize.
Tesseractのunicharsetファイルには、Tesseract OCRエンジンが認識するように訓練された各シンボル（unichar）に関する情報が含まれています。

Currently, generating the unicharset file is done in two steps using these commands:
現在、unicharsetファイルの生成は、次のコマンドを使用して2つのステップで行われます。

```
unicharset_extractor and(と) set_unicharset_properties.
```

NOTE: The unicharset file must be regenerated whenever inttemp, normproto and pffmtable are generated (i.e. they must **all** be recreated when the box file is changed) as they have to be in sync.
注： inttemp、normproto、およびpffmtableが生成されるたびに（つまり、ボックスファイルが変更されたときにすべて再作成される必要があるため）、unicharsetファイルは再生成される必要があります。

For more details about the unicharset file format, see this appendix.
unicharsetファイル形式の詳細については、この（巻末の）付録を参照してください。

unicharset_extractor

Tesseract needs to know the set of possible characters it can output. To generate the unicharset data file, use the unicharset_extractor program on the box files generated above:
Tesseractは、出力可能な文字のセットを知る必要があります。 unicharsetデータファイルを生成するには、上記で生成されたボックスファイルに対してunicharset_extractorプログラムを使用します。

```
unicharset_extractor lang.fontname.exp0.box lang.fontname.exp1.box ...
```

set_unicharset_properties

New in 3.03 3.03の新機能

This tool, together with a set of data files, allow the addition of extra properties in the unicharset, mostly sizes obtained from fonts.
このツールは、一連のデータファイルと一緒に、ほとんどの場合フォントから取得されたサイズを、ユニキャストに追加のプロパティを追加することを可能にします。

```
training/set_unicharset_properties -U input_unicharset -O
output_unicharset --script_dir=training/langdata
```

--script-dir should point to a directory containing the relevant .unicharset file(s) for your training character set. These can be downloaded from https://github.com/tesseract-ocr/langdata).
--script-dirは、トレーニングキャラクタセットに関連する.unicharsetファイルを含むディレクトリを指す必要があります。これらはhttps://github.com/tesseract-ocr/langdataからダウンロードできます。

After running unicharset_extractor and set_unicharset_properties, you should get a unicharset file with all the fields set to the right values, like in this example.
unicharset_extractorとset_unicharset_propertiesを実行した後は、この例のように、すべてのフィールドが正しい値に設定されたunicharsetファイルを取得するはずです。

The font_properties file font_propertiesファイル

Now you need to create a font_properties text file. The purpose of this file is to provide font style information that will appear in the output when the font is recognized.
今度はfont_propertiesテキストファイルを作成する必要があります。このファイルの目的は、フォントが認識されたときに出力に表示されるフォントスタイル情報を提供することです。

Each line of the font_properties file is formatted as follows:
font_propertiesファイルの各行は、次のようにフォーマットされています。

```
fontname italic bold fixed serif fraktur
```

where fontname is a string naming the font (no spaces allowed!), and italic, bold, fixed, serif and fraktur are all simple 0 or 1 flags indicating whether the font has the named property.
ここで、fontnameはフォントの名前を表す文字列（スペースは不可）、italic、bold、fixed、serif、frakturはすべて、フォントが名前付きプロパティを持つかどうかを示す単純な0または1のフラグです。

Example: **例**:

```
timesitalic 1 0 0 1 0
```

The font_properties file will be used by the shapeclustering and mftraining commands.
font_propertiesファイルはshapeclusteringおよびmftrainingコマンドによって使用されます。

When running mftraining, each fontname field in the *.tr file must match an fontname entry in the font_properties file, or mftraining will abort.
mftrainingを実行するとき、*。trファイルの各fontnameフィールドはfont_propertiesファイルのfontnameエントリと一致しなければなりません。そうしないと、mftrainingは中止されます。

Note: There is a default font_properties file, that covers 3000 fonts (not necessarily accurately) located in the langdata repository.
注: langdataリポジトリにある3000個のフォント（必ずしも正確ではない）をカバーするデフォルトのfont_propertiesファイルがあります。

Clustering クラスタリング

When the character features of all the training pages have been extracted, we need to cluster them to create the prototypes.
すべてのトレーニングページの特徴が抽出されたら、それらをクラスタ化してプロトタイプを作成する必要があります。

The character shape features can be clustered using the shapeclustering, mftraining and cntraining programs:
シェイパークラスタリング、mftraining、および cntraining プログラムを使用して、キャラクタシェイプフィーチャをクラスタ化できます。

shapeclustering

shapeclustering **should not normally be used except for the Indic languages.** (It may be necessary to use shapeclustering for Latin-alphabet languages where base characters are combined with two or more non-spacing modifiers, e.g. e̋, ő, r̥̄: this includes, but may not be limited to, Indic languages in Latin transcription. Failure to do so will lead mftraining to give an Assert failed:in file unicharset.cpp error.)
shaのクラスタ化は通常、インド語以外の言語では使用しないでください。（ベース文字が2つ以上の非スペーシング修飾子と組み合わされているラテン系アルファベット言語では、シャープクラスタリングを使用する必要があるかもしれません。例えば、e̋、、、rなど。そうしないと、mftrainingがAssertを失敗させることになります。in file unicharset.cppエラー）

```
shapeclustering -F font_properties -U unicharset lang.fontname.exp0.tr
lang.fontname.exp1.tr ...
```

shapeclustering creates a master shape table by shape clustering and writes it to a file named shapetable.
shapeclustering は、シェイプクラスタリングによってマスターシェイプテーブルを作成し、それを shapetable という名前のファイルに書き込みます。

mftraining

```
mftraining -F font_properties -U unicharset -O lang.unicharset
lang.fontname.exp0.tr lang.fontname.exp1.tr ...
```

The -U file is the unicharset generated by unicharset_extractor above, and lang.unicharset is the output unicharset that will be given to combine_tessdata.
-Uファイルは上記のunicharset_extractorによって生成されたunicharsetであり、lang.unicharsetはcombine_tessdataに与えられる出力unicharsetです。

mftraining will output two other data files: inttemp (the shape prototypes) and pffmtable (the number of expected features for each character).
mftrainingは他の2つのデータファイルを出力します：inttemp（形状プロトタイプ）とpffmtable（各文字に期待される機能の数）。

NOTE: mftraining will produce a shapetable file if you didn't run shapeclustering. You **must** include this shapetable in your traineddata file, whether or not shapeclustering was used.
注： shapeclustering を実行しなかった場合、mftraining はシェイプテーブルファイルを作成します。シェイプクラスタリングが使用されているかどうかにかかわらず、このシェイプテーブルをあなたの学習されたデータファイルに含める必要があります。

cntraining

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

This will output the normproto data file (the character normalization sensitivity prototypes).
これはnormprotoデータファイル（文字正規化感度プロトタイプ）を出力します。

Dictionary Data (Optional) 辞書データ（オプション）

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.
Tesseractは各言語につき最大8つの辞書ファイルを使用します。これらはすべてオプションであり、Tesseractがさまざまな文字の組み合わせの可能性を判断するのに役立ちます。

Seven of the files are coded as a Directed Acyclic Word Graph (DAWG), and the other is a plain UTF-8 text file:
7つのファイルは有向非循環ワードグラフ（DAWG）としてコード化されており、もう1つはプレーンなUTF-8テキストファイルです。

Name **名前**	**Type** **タイプ**	**Description** **説明**
word-dawg	dawg	A dawg made from dictionary words from the language. 言語からの辞書の単語から作られたdawg。
freq-dawg	dawg	A dawg made from the most frequent words which would have gone into word-dawg. 最も頻繁に使われる単語から作られた単語。
punc-dawg	dawg	A dawg made from punctuation patterns found around words. The *"word"* part is replaced by a single space. 単語のまわりに見られる句読パターンから作られたdawg。「単語」の部分は単一のスペースに置き換えられます。
number-dawg	dawg	A dawg made from tokens which originally contained digits. Each digit is replaced by a space character. もともと数字を含んでいたトークンから作られたdawg。各桁はスペース文字に置き換えられます。
bigram-dawg	dawg	A dawg of word bigrams where the words are separated by a space and each digit is replaced by a *?*. 単語がスペースで区切られ、各数字が？で置き換えられている単語bigramsのdawg。
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). 辞書に追加する追加の単語のリスト。ユーザーが必要に応じて追加するために、通常は空のままにします。 tesseract（1）を参照してください。 (https://github.com/tesseract-ocr/tesseract/blob/13b7900ebf21fbccbc3d89ebf63cc7165b6ae2ca/doc/tesseract.1.asc#config-files-and-augmenting-with-user-data)

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, aspell or hunspell) - 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:

DAWG辞書ファイルを作成するには、まずあなたの言語のワードリストが必要です。スペルチェッカの単語リスト（ispell、aspell、hunspellなど）の基礎として使用する適切な辞書ファイルを見つけることができます - ライセンスについては注意してください。ワードリストは、1行に1ワードのUTF-8テキストファイルとしてフォーマットされています。単語リストを必要なセット、例えば、頻繁な単語と残りの単語に分割してから、wordlist2dawgを使用してDAWGファイルを作成します。

```
wordlist2dawg frequent_words_list lang.freq-dawg lang.unicharset
wordlist2dawg words_list lang.word-dawg lang.unicharset
```

For languages written from right to left (RTL), like Arabic and Hebrew, add -r 1 to the wordlist2dawg command.
アラビア語やヘブライ語など、右から左に書かれた言語（RTL）の場合は、wordlist2dawgコマンドに-r 1を追加します。

Other options can be found in wordlist2dawg Manual Page
他のオプションはwordlist2dawgマニュアルページにあります。
(https://github.com/tesseract-ocr/tesseract/blob/master/doc/wordlist2dawg.1.asc)

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.
注： 辞書ファイルが結合トレーニングデータに含まれている場合は、少なくとも1つのエントリが含まれている必要があります。それ以外の場合は空になる辞書ファイルは、combine_tessdataステップには必要ありません。

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)
スペルが異常な単語は辞書ファイルに追加する必要があります。異常なスペルには、英字と句読点または数字の混在が含まれる場合があります。（例：国際化、国際化、国際化、google.com、news.bbc.co.uk、io9.com、utf8、ucs2）

If you need example files for dictionary wordlists, uncombine (with combine_tessdata) existing language data file (e.g. eng.traineddata) and then extract wordlist with dawg2wordlist
辞書の単語リスト用のサンプルファイルが必要な場合は、（combine_tessdataを使用して）既存の言語データファイル（例：eng.traineddata）を結合解除してから、dawg2wordlistを使用して単語リストを抽出します。

The unicharambigs file unicharambigsファイル

The unicharambigs file is a text file that describes possible ambiguities between characters or sets of characters, and is manually generated.
unicharambigsファイルは、文字間または文字セット間のあいまいさを説明するテキストファイルで、手動で生成されます。

To understand the file format, look at the following example:
ファイル形式を理解するために、次の例を見てください。

```
v1
2       ' '     1       "       1
1       m       2       r n     0
3       i i i   1       m       0
```

The first line is a version identifier.
1行目はバージョンIDです。

The remaining lines are **tab** separated fields, in the following format:
残りの行はタブ区切りのフィールドで、次の形式です。

```
<number of characters for match source> <characters for match source> <number of characters for match target> <characters for match target> <type indicator>
```

Type indicator could have following values:
タイプ標識は以下の値を取ります。

Value **値**	**Type** **タイプ**	**Description** **説明**
0	A non-mandatory substitution. 必須ではない置換。	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. これは、 'source'を 'target'に置き換えることで辞書以外の単語から辞書の単語が作成された場合でも、あいまいさがセグメンテーション検索へのヒントとして役立つことを意味します。あいまいさを介して別の辞書の単語に変えることができる辞書の単語は、適応分類器を訓練するためには使用されません。
1	A mandatory substitution. 必須の代替です。	This informs tesseract to always replace the matched 'source' with the 'target' strings. これは、一致した「source」を常に「target」の文字列に置き換えるようにtesseractに通知します。

Example line **行の例**	**Explanation** **説明**
`2 ' ' 1 " 1`	A double quote (") should be substituted **whenever** 2 consecutive single quotes (') are seen. 2つの連続した単一引用符 (') が見られるときは**いつでも**、二重引用符 (") を代用する必要があります。
`1 m 2 r n 0`	The characters 'rn' may sometimes be recognized incorrectly as 'm'. 文字 'rn' は 'm' として誤って認識されることがあります。
`3 i i i 1 m 0`	The character 'm' may sometimes be recognized incorrectly as the sequence 'iii'. 文字 'm' は、シーケンス 'iii' として誤って認識されることがあります。

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.
それぞれの別々の文字は、unicharsetに含まれていなければなりません。つまり、使用されるすべての文字は、訓練されている言語の一部でなければなりません。

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.
規則は双方向ではないため、「m」が検出されたときに「rn」を考慮したい場合、およびその逆の場合は、それぞれに規則が必要です。

Version 3.03 and on supports a new, simpler format for the unicharambigs file:
バージョン3.03以降では、unicharambigsファイル用の新しいよりシンプルなフォーマットがサポートされています。

```
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.
このフォーマットでは、 "error"と "correct"はスペースで区切られた単純なUTF-8文字列であり、スペースの後にはv1と同じ型指定子（オプションの場合は0、必須の置換の場合は1）。この単純な形式の欠点は、TesseractがUTF-8文字列をunicharsetのコンポーネントにエンコードする必要があることです。複雑なスクリプトでは、このエンコードはあいまいな場合があります。この場合、エンコーディングは、各コンポーネントにUTF-8の最小文字を使用するように選択されます。つまり、最短のユニキャストコンポーネントがエンコーディングを構成します。

Like most other files used in training, the unicharambigs file must be encoded as UTF-8, and must end with a newline character.
トレーニングで使用される他のほとんどのファイルと同様に、unicharambigsファイルはUTF-8としてエンコードされ、改行文字で終わらなければなりません。

The unicharambigs format is also described in the unicharambigs(5) man page.
unicharambigsフォーマットは、unicharambigs（5）のマニュアルページにも記載されています。

The unicharambigs file may also be non-existent.
unicharambigsファイルも存在しない可能性があります。

Putting it all together すべてを一緒に入れて

That is all there is to it! All you need to do now is collect together all the files (shapetable, normproto, inttemp, pffmtable, unicharset) and rename them with a lang. prefix (for example eng.), and then run combine_tessdata on them as follows:
それだけです。今する必要があるのは、すべてのファイル（shapetable、normproto、inttemp、pffmtable、unicharset）をまとめて、それらをlangで名前変更することです。接頭辞（たとえばeng。）を使用してから、次のようにそれらに対してcombine_tessdataを実行します。

```
combine_tessdata lang.
```

Although you can use any string you like for the language code, we recommend that you use a 3-letter code for your language matching one of the ISO 639-2 codes.
言語コードには任意の文字列を使用できますが、ISO 639-2コードのいずれかに一致する言語には3文字のコードを使用することをお勧めします。

The resulting lang.traineddata goes in your tessdata directory. Tesseract can then recognize text in your language (in theory) with the following:
結果のlang.traineddataはあなたのtessdataディレクトリに入ります。 Tesseractはそれからあなたの言語のテキストを（理論的に）次のように認識することができます：

```
tesseract image.tif output -l lang
```

More options of combine_tessdata can be found on its Manual Page or in comment of its source code.
combine_tessdataのその他のオプションは、そのマニュアルページまたはそのソースコードのコメントにあります。

You can inspect some of the internals of traineddata files in 3rd party online Traineddata inspector.
あなたは第三者のオンライン訓練データ検査官で訓練データファイルの内部のいくつかを調べることができます。 (https://te-traineddata-ui.herokuapp.com/)

Appendices 付録

The *.tr file format * .trファイル形式

Every character in the box file has a corresponding set of entries in the .tr file (in order) like this:
ボックスファイル内のすべての文字は、次のように.trファイル内に対応するエントリのセットを（順番に）持っています。

```
<fontname> <character> <left> <top> <right> <bottom> <pagenum>
 4
mf <number of features>
<x> <y> <length> <dir> 0 0
...
cn 1
<ypos> <length> <x2ndmoment> <y2ndmoment>
if <number of features>
<x> <y> <dir>
...
tb 1
<bottom> <top> <width>
```

The Micro Features (mf) are polygon segments of the outline normalized to the 1st and 2nd moments. The mf line will followed by a set of lines determined by <number of features>.
Micro Features(mf)は、1次モーメントと2次モーメントに正規化されたアウトラインのポリゴンセグメントです。 mf行の後には、<number of features>によって決定される一連の行が続きます。
x is x position [-0.5,0.5]　xはx位置です[-0.5,0.5]
y is y position [-0.25,0.75]　yはy位置です[-0.25,0.75]
length is the length of the polygon segment [0,1.0]　lengthはポリゴンセグメントの長さです[0,1.0]
dir is the direction of the segment [0,1.0]　dirはセグメントの方向です[0,1.0]

The Char Norm features (cn) are used to correct for the moment normalization to distinguish position and size (eg c vs C and , vs ')
Char Norm機能(cn)は、位置とサイズを区別するためのモーメント正規化を補正するために使用されます（例えば、c vs Cと、vs '）。

if - Int Features　if - Intの機能

tb - Geo features　tb - ジオ機能

The unicharset file format unicharsetファイルフォーマット

Tesseract's unicharset file contains information on each symbol (unichar) the Tesseract OCR engine is trained to recognize.
Tesseractのunicharsetファイルには、Tesseract OCRエンジンが認識するように訓練された各シンボル（unichar）に関する情報が含まれています。

The first line of a unicharset file contains the number of unichars in the file.
unicharsetファイルの最初の行には、ファイル内のunicharsの数が含まれています。

After this line, each subsequent line provides information for a single unichar. The first such line contains a placeholder reserved for the space character.
この行の後に続く各行は、単一のunicharに関する情報を提供します。そのような最初の行は、スペース文字のために予約されたプレースホルダを含みます。

Each unichar is referred to within Tesseract by its Unichar ID, which is the line number (minus 1) within the unicharset file. Therefore, space gets unichar 0.
各unicharはTicseract内ではそのUnichar IDによって参照されます。これは、unicharsetファイル内の行番号（マイナス1）です。したがって、スペースは0になります。

Each unichar line in the unicharset file should have these space-separated fields:
unicharsetファイルの各unichar行には、次のスペース区切りのフィールドがあります。

character properties glyph_metrics script other_case direction mirror normed_form

- character キャラクター
  The UTF-8 encoded string to be produced for this unichar.
  このunichar用に生成されるUTF-8エンコードの文字列
- properties プロパティ
  An integer mask of character properties, one per bit. From least to most significant bit, these are: isalpha, islower, isupper, isdigit, ispunctuation.
  文字プロパティの整数マスク。ビットごとに1つ。最下位から最上位まで、isalpha、islower、isupper、isdigit、ispunctuationです。
- glyph_metrics グリフメトリクス
  Ten comma-separated integers representing various standards for where this glyph is to be found within a baseline-normalized coordinate system where 128 is normalized to x-height.
  128がx-heightに正規化されているベースライン正規化座標系内でこのグリフが見つかる場所のさまざまな標準を表す10個のコンマ区切りの整数。
  - min_bottom, max_bottom
    The ranges where the bottom of the character can be found.
    文字の下端が見つかる範囲。
  - min_top, max_top
    The ranges where the top of the character may be found.
    文字の先頭が見つかる可能性がある範囲。
  - min_width, max_width
    Horizontal width of the character. 文字の横幅
  - min_bearing, max_bearing
    How far from the usual start position does the leftmost part of the character begin.
    通常の開始位置からキャラクタの左端部分までの距離。
  - min_advance, max_advance
    How far from the printer's cell left do we advance to begin the next character.
    プリンタのセルからどれだけ離れているか、次の文字から始めます。
- script スクリプト
  Name of the script (Latin, Common, Greek, Cyrillic, Han, NULL).
  スクリプトの名前（ラテン語、共通文字、ギリシャ語、キリル文字、漢字、NULL）。
- other_case 他のケース
  The Unichar ID of the other case version of this character (upper or lower).
  この文字の他のケースバージョンのUnichar ID（上限または下限）。

- direction 方向
  The Unicode BiDi direction of this character, as defined by ICU's enum UCharDirection. (0 = Left to Right, 1 = Right to Left, 2 = European Number…)
  この文字のUnicode BiDi方向。ICUのenum UCharDirectionで定義されています。（0 =左から右、1 =右から左、2 =ヨーロッパの数字…）
- mirror ミラー
  The Unichar ID of the BiDirectional mirror of this character. For example the mirror of open paren is close paren, but Latin Capital C has no mirror, so it remains a Latin Capital C.
  このキャラクタの双方向ミラーのUnichar ID。たとえば、開いた親の鏡は親の親ですが、ラテン大文字Cには鏡がないため、ラテン大文字Cのままです。
- normed_form
  The UTF-8 representation of a "normalized form" of this unichar for the purpose of blaming a module for errors given ground truth text. For instance, a left or right single quote may normalize to an ASCII quote.
  グランドトゥルーステキストを与えられたエラーのためにモジュールを非難する目的のためのこのunicharの「正規化された形式」のUTF-8表現。たとえば、左または右の一重引用符はASCII引用符に正規化できます。

An example of the unicharset file unicharset ファイルの例

```
110
NULL 0 NULL 0
N 5 59,68,216,255,87,236,0,27,104,227 Latin 11 0 1 N
Y 5 59,68,216,255,91,205,0,47,91,223 Latin 33 0 2 Y
1 8 59,69,203,255,45,128,0,66,74,173 Common 3 2 3 1
9 8 18,66,203,255,89,156,0,39,104,173 Common 4 2 4 9
a 3 58,65,186,198,85,164,0,26,97,185 Latin 56 0 5 a
...
```

More about the properties field 追補・properties フィールドについて

Here is another example. For simplicity only the first two fields in each line are shown in this example. The other fields are omitted.
これは別の例です。簡単にするために、この例では各行の最初の2つのフィールドだけを示しています。他のフィールドは省略されています。

```
...
; 10 ...
b 3 ...
W 5 ...
7 8 ...
= 0 ...
...
```

Char 文字	Punctuation 句読点	Digit 数字	Upper 上位	Lower 下位	Alpha アルファ	Binary num 2進数	Hex. 16進数
;	1	0	0	0	0	10000	10
b	0	0	0	1	1	00011	3
W	0	0	1	0	1	00101	5
7	0	1	0	0	0	01000	8
=	0	0	0	0	0	00000	0

In columns 2-6, 0 means 'No', 1 means 'Yes'.
2から6桁目で、0は「いいえ」を意味し、1は「はい」を意味します。

Expect to add LCD font library。！　LCDフォントライブラリを追加する予定です。