# Document Bachelor Thesis Clarification

**Name: Chuxuan Li** **Date: 27.02.2018**

ZD AUTOMOTIVE
Focus on Technology Shaping the Future

# Table of Contents

# 1. Program for text recognition

The program “Fehlererkennung.py” can be run in Terminal. Before running:

1. Make sure the first line in program is: `#!/usr/bin/python`
2. In terminal: `chmod a+x Fehlererkennung.py`
3. Run the Python file: `./Fehlererkennung.py [1] [2] [3]`
   [1] screenshots folder source, [2] xml file source, [3] name of “traineddata”

E.g.: `./Fehlererkennung.py 'test/' 'test/CAR_DRIVE_SELECT_INDIVIDUAL_Evo2plus.xml' 't8'`

Another program “Texterkennung.py” can also be run in terminal: `./Texterkennung.py [1] [2]`
[1] screenshots folder source, [2] name of “traineddata”
E.g.: `./Texterkennung.py 'test/' 't8'`

In the program there are 8 functions defined. “load_screenshots”, “image_processing” and “filter_screenshots” are related to image processing and text recognition. And the last 5 functions are defined for comparison of actual and target texts. In program “Texterkennung.py” the last 5 functions are commented und don’t work.

```
import ...

def load_screenshots(img_path, images):...

def image_processing(imageA, img0, lang, write_file):...

def filter_screenshots(images, ref, lang, img_info_file):...

def get_target_value(xml_file, target_value_file):...

def get_actual_elements(imgInfoFile, actualElementFile):...

def get_actual_value(actual_element_file, actual_value_file):...

def compare_difference(target_value_file, actual_value_file):...

def compare_difference2(target_value_file, actual_value_file):...

def main():...

main()
```

## 1.1 Packages

```
import csv
import cv2 # 1
import glob
from PIL import Image
from skimage.measure import compare_ssim as ssim
import sys
import tesserocr # 2
import time
import xml.etree.ElementTree as ET
```

1. Package of OpenCV 2
2. Package of Tesseract-Engine for Python

## 1.2 Read images

**Under function load_screnshots()**

```
def load_screenshots(img_path, images):
    filenames = [img for img in glob.glob(img_path)] # 1
    filenames.sort() # 2
    for img in filenames:
        n = cv2.imread(img) # 3
        images.append(n)
```

1. the filenames in the folder are read into the list filenames randomly
2. the filenames are sorted in order
3. cv2.imread to read the images, cv2.imwrite to write images

## 1.3 Filter repeated images (Wiederholungsanalyse)

**Under function filter_screenshots(),**
**call function image_processing()**

1. Cut screenshot, only chinese part remains
2. Turn screenshots into grayscale images
3. Resize the images in to (8, 8)

4. compare ssim differences between image i and image i+1
5. If the ssim similarity between image i and image i+1 is larger than 0.99, do image_processing for image i and skip image i+1, compare image i and image i+2

## 1.4 Image processing (Bildverarbeitung)

### Under function image_processing()

### 1.4.1 Threshold

```
def image_processing(imageA, img0, lang, csv_file):
    count = 0
    img = imageA.copy()
    # prepare image quality for OCR
    img = cv2.bitwise_not(img) # 1
    _, img_recog = cv2.threshold(img, 210, 255,
cv2.THRESH_BINARY) # 2
    _, img = cv2.threshold(img, 224, 255, cv2.THRESH_BINARY)
    #find text areas
    imgBi = cv2.bitwise_not(imageA)
    _, binary2 = cv2.threshold(imgBi, 0, 255,
cv2.THRESH_BINARY)
```

1. Grayscale has a value in range of (0,255). “cv2.bitwise_not” reverse the color of the image, white to black, black to white (new grayscale value = 255 - old value).
2. “cv2.threshold” changes the grayscale image into binary image with only black and white, no more gray with different grayscale value. The second parameter sets the threshold value. In the program threshold are carried out three times:
    a) Value 210 for title part [104:204, 319:1493]([y:y+h, x:x+w]) to remain the texts with color, e.g. 车辆 in red
    b) Value 0 for other interest part, **if** x2 > 120 **and** y2 > 200 **and** 2 < w2 **and** 2 < h2 < 450. The value o removes most information in the image, including most of the pixels of texts. The target here is, together with following steps to find the text areas.
    c) Value 224 for all interest part. Remove the background as much as possible, remain pixels of the texts as much as possible. After the coordinates of text areas are found, every

text area in the processed image “img” will be extracted for text recognition.


Processing effect at threshold value 50


Processing effect at threshold value 220

**Note:** Some parts in the screenshots cannot be removed even when the threshold value is 0. They will sometimes recognized as texts. When the text areas can be located through other ways instead of through “find contours”, the recognition result would be better.

**Note:** The program can only maintain the texts in color for title area. When colorful texts show in main area of the screenshots, they will currently be filtered through “threshold”.

### 1.4.2 Find contours

```
kernel = cv2.getStructuringElement(cv2.MORPH_RECT, (21, 20))
eroded = cv2.erode(binary2, kernel, iterations=1) # 1
erodedBi = cv2.bitwise_not(eroded)
contours2, hierarchy2 = cv2.findContours(erodedBi,
cv2.RETR_EXTERNAL, cv2.CHAIN_APPROX_SIMPLE) # 2
```

1. The steps above defines the erosion process. (21, 20) defines the size of the erosion kernel. The neighbor pixels of one black pixel in the range of this kernel will all turned into black. Through erosion "cv2.erode" the neighbor characters in the image are connected.
2. "cv2.findContours" find the contours of all connected areas.

**Note:** The size of the kernel is general for all screenshots. In some cases the texts are connected with icons. In some cases, when the texts is a paragraph, this paragraph will be recognized as many lines instead of one area.

## 1.5 Text recognition

**Under function image_processing(),**
**call Tesseract for recognition**

```
for j in range(len(contours2)):
      cnt2 = contours2[j]
      x2, y2, w2, h2 = cv2.boundingRect(cnt2) # 1
      if x2 > 120 and y2 > 200 and 2 < w2 and 2 < h2 < 450:
         count += 1
         cv2.rectangle(img0, (x2, y2), (x2+w2, y2+h2), (0,
255, 0), 2) # 3
         crop_img = img_recog[y2:y2+h2, x2:x2+w2]
         cv2.imwrite('ref.png', crop_img) # 2
         text =
tesserocr.image_to_text(Image.open('ref.png'), lang) # 3
         text = text.replace(" ", "") # 4
         text = text.replace("\n", " ") # 5
         csv_file.write('{}:,{},{},{},{},{}\n'.format\
(count, x2, y2, w2, h2, text.encode('utf-8'))) # 6
      else:
         pass
```

1. Get the coordinates of the current contour
2. For every contour found, the text area will be extracted with the coordinates information and generate a new temporary image “ref.png”
3. “tesserocr.image_to_text” recognizes the texts from “ref.png”, the parameter “lang” refers to the “traineddata” that Tesseract calls.
4. In some recognition results there are unexpected blank spaces between two characters. All the blank spaces are removed. This step only applies to chinese. When the text is a combination of english and chinese, it will influence the comparison result. E.g. “Audi 中国” in the recognition result after removing blank space is not the same as “Audi 中国” from the xml file.
5. When there is a wrap “\n” in the result of a text area, it will also be removed, so that the result can be saved together with the coordinates information together in one line.
6. The recognition result of every contour will be written as one line into the output file.

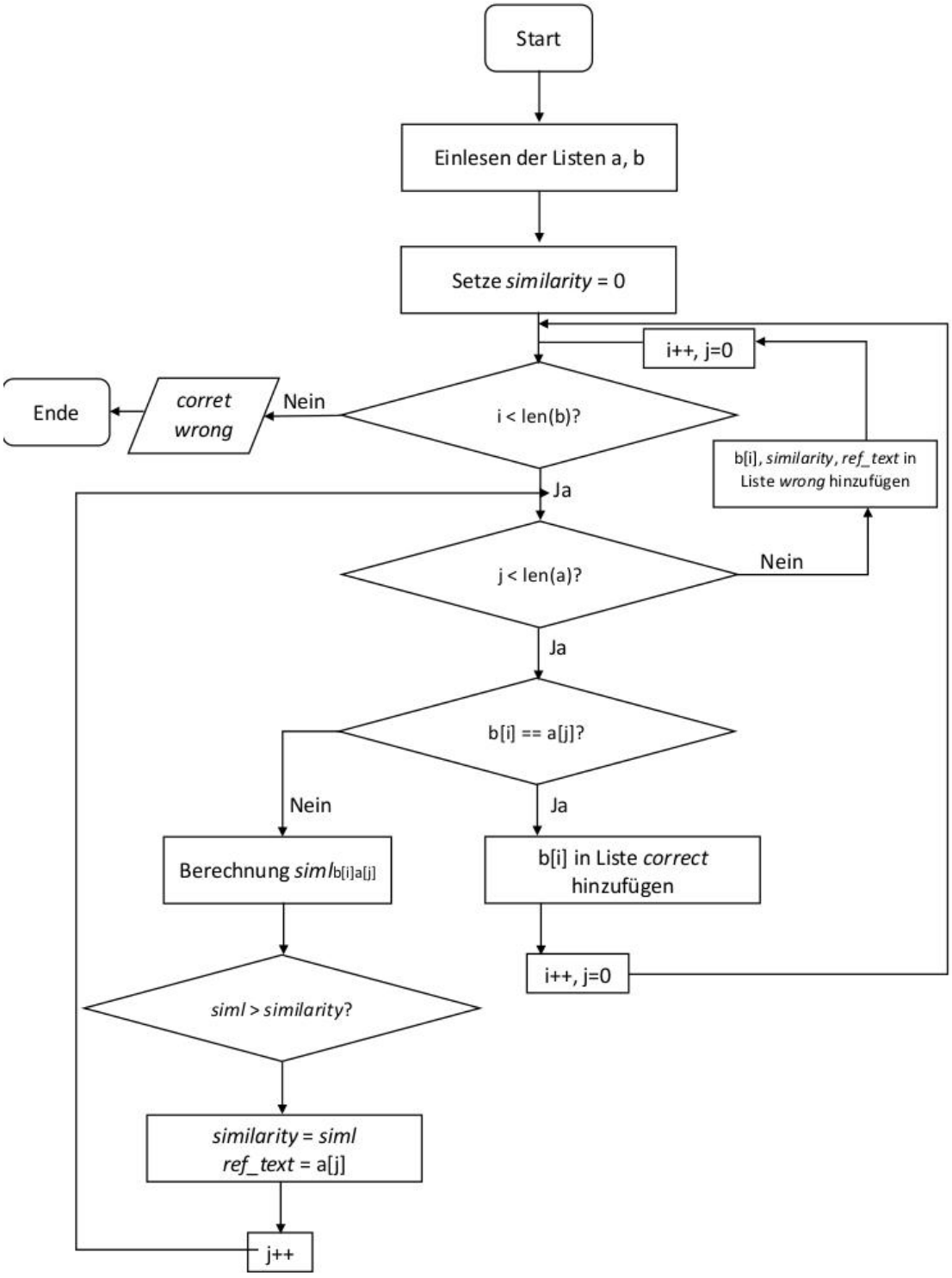
## 1.6 Compare difference of actual texts and target texts (Fehlererkennung)

**Under function:**
**get_target_value(),**
**get_actual_elements(),**
**get_actual_value(),**
**compare_difference(),**
**compare_difference2()**

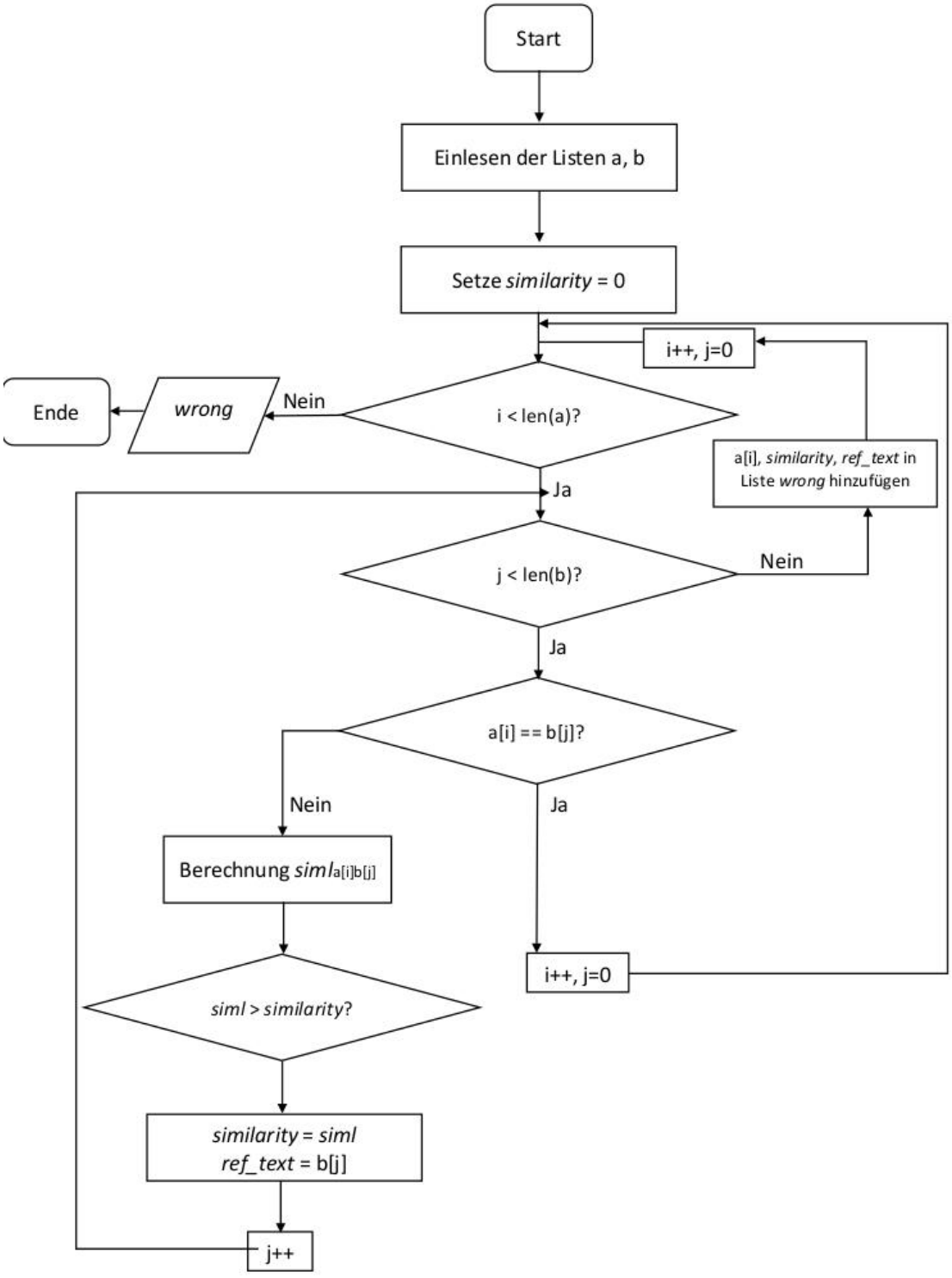
The actual and target value of texts will be extracted and read into two lists a & b.

The comparison process is carried out in 2 directions:

a) actual texts as reference
b) target texts as reference




actual texts as reference




target texts as reference

The texts will first be checked if they are the same. If not, the strings will be split into separate chinese characters. The similarity between actual and target texts will be counted: how many common chinese characters there are.

# 2. Training of Tesseract ‘traineddata’

## 2.1 Background

- Tesseract Version: 3.04
- Official documentation of training:

https://github.com/tesseract-ocr/tesseract/wiki/Training-Tesseract

- additional libraries are required to build the training tools:

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

- make and install the training tools with separate make commands:

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

- Font file path: /usr/share/fonts
- Font name of chinese text: ‘ZYHei_GB18030_c’

## 2.2 Training Procedure

An introduction in Chinese:
http://qianjiye.de/2015/08/tesseract-ocr

### 2.2.1 Prepare training text

There is no official documentation of how the training text for Chinese should be organized. Some regularities found are concluded as follows:
- the text should consist of random combination of characters
- Although don’t know why, Tesseract seems not to perform well to extract features from single character in the training text (there is a blank space between every two characters)

- Tesseract doesn’t work well neither, when there is totally no blank space in the training set. Tesseract will split a character into two parts at recognition.
- For Latin languages it is expected that there are a minimum number of samples of each character. For example, 10 ‘g’ and 13 ‘a’ in the training text. However, in Chinese training, there’s no big difference in the accuracy of recognition results, when every Chinese character appears only once or some characters appears many times. When the repetition of one character is to high, it even results in an overfitting of the features of one character.

So the training text is organized like this:

1. Extract the expected texts from the xml file
2. Use online tools to calculate the character frequency of the texts, so that a character list without repeat character is available after processing the output file (tool url: http://www.aihanyu.org/cncorpus/CpsTongji.aspx)
3. Randomly add blank spaces among characters, basically every two or three characters

When new characters need to be added for training, they should be added at the end of the training texts in the data, so that the order and features of former old characters won’t be changed and the features can stay stable at training.

### 2.2.2 Generate 'tif/box' file

From this step on the training is executed by Tesseract through terminal.

**(if needed)** list all available fonts to make sure that the font file is already in the default font file path for the system:

```
text2image --list_available_fonts --fonts_dir=/usr/share/fonts
```

Two files will be created in this step:

- ‘t9.ZYHei_GB18030_c.exp0.tif’, image file
- ‘t9.ZYHei_GB18030_c.exp0.box’, box file with coordinates information

```
text2image --text=training_text.txt --outputbase=t9.ZYHei_GB18030_c.exp0
--font='ZYHei_GB18030_c' --fonts_dir=/usr/share/fonts
```

### 2.2.3 Run tesseract for training - generate '.tr' file (feature file 1)

```
tesseract t9.ZYHei_GB18030_c.exp0.tif t9.ZYHei_GB18030_c.exp0 box.train
```

The detail description of the features in this file can be seen in the following image:

**The *.tr file format**

Every character in the box file has a corresponding set of entries in the .tr file (in order) like this:

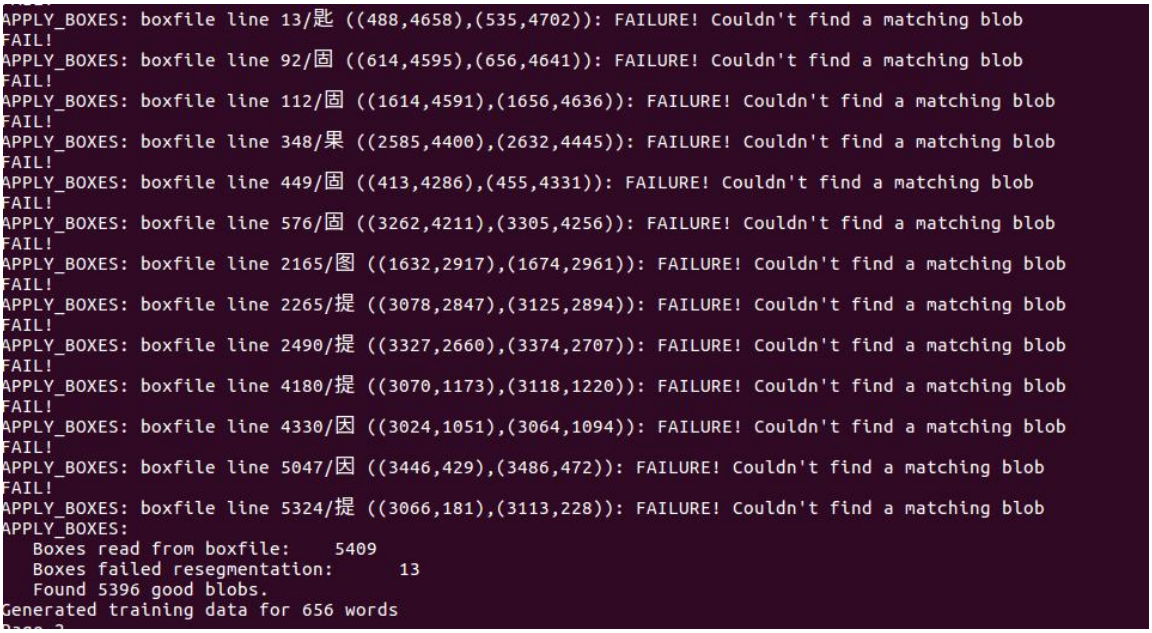
```
<fontname> <character> <left> <top> <right> <bottom> <pagenum>
 4
mf <number of features>
<x> <y> <length> <dir> 0 0
...
cn 1
<ypos> <length> <x2ndmoment> <y2ndmoment>
if <number of features>
<x> <y> <dir>
...
tb 1
<bottom> <top> <width>
```

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>.
x is x position [-0.5,0.5]
y is 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 Char Norm features ( `cn` ) are used to correct for the moment normalization to distinguish position and size (eg `c` vs `C` and `,` vs `'` )

The mf features in the '.tr' file will be clustered in step 5) Clustering 1.
The cn features will be clustered in step 6) Clustering 2.

Sometimes after training Tesseract will give warnings like this:




It means the features of these characters are not successfully extracted. In these cases we should adjust the training text file according to the warnings and then repeat step 1) and 2), until the warnings maintain at a minimum level.

**Note:** when adjusting the training text file, it's recommended to exchange the position of the warning character and its neighbor character, so that the position of other characters stay the same and the feature extraction of other characters stays stable.

**Note:** when a character appears more than once in the training text file, but this character appears only once in the warning text, then it's acceptable. Tesseract also get its features. But at best the warning of this single character can also be solved.

### 2.2.4 Unicharset_extractor - generate 'unicharset' (feature file 2)

a) create the 'unicharset' file with all values default

```
unicharset_extractor t9.ZYHei_GB18030_c.exp0.box
```

b) create the 'output_unicharset' file. The default value in 'unicharset' file will be changed into actual value.

```
set_unicharset_properties -U unicharset -O output_unicharset --script_dir =
~/Dokumente/Bachelorarbeit_Chu/Texttool/T/*.unicharset
```

The detail description of the features in this file are as follows (8 types of features, 17 values in total for one character):

`character` `properties` `glyph_metrics` `script` `other_case` `direction` `mirror` `normed_form`

- `character`
  The UTF-8 encoded string to be produced for this unichar.
- `properties`
  An integer mask of character properties, one per bit. From least to most significant bit, these are: isalpha, islower, isupper, isdigit, ispunctuation.
- `glyph_metrics`
  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.
  - `min_bottom`, `max_bottom`
    The ranges where the bottom of the character can be found.
  - `min_top`, `max_top`
    The ranges where the top of the character may be found.
  - `min_width`, `max_width`
    Horizontal width of the character.
  - `min_bearing`, `max_bearing`
    How far from the usual start position does the leftmost part of the character begin.
  - `min_advance`, `max_advance`
    How far from the printer's cell left do we advance to begin the next character.
- `script`
  Name of the script (Latin, Common, Greek, Cyrillic, Han, NULL).
- `other_case`
  The Unichar ID of the other case version of this character (upper or lower).
- `direction`
  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…)
- `mirror`
  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.
- `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.

However, there is a problem here which is still open: the values of the ten features from 'glyph_metrics' cannot be written into the file 'output_unicharset', which means this feature file is incomplete. This file will be used at step 5) Clustering 1. At clustering the features are incomplete, so this is a potential cause for the errors of recognition.

### 2.2.5 Create 'font_properties' text file

This file is created manually. It tells Tesseract the properties of the selected font. In this case all values of the properties of ‘ZYHei_GB18030_c’ are set as 0.

```
echo ZYHei_GB18030_c 0 0 0 0 0 0 >font_properties
```

## The font_properties file

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.

Each line of the `font_properties` file is formatted as follows: `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.

**Example**:

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

The `font_properties` file will be used by the `shapeclustering` and `mftraining` commands.

When running `mftraining`, each `fontname` field in the *.tr file must match an `fontname` entry in the `font_properties` file, or `mftraining` will abort.

**Note**: There is a default font_properties file, that covers 3000 fonts (not necessarily accurately) located in the langdata repository.

### 2.2.6 Clustering 1 - generate 'shapetable', 'inttemp', 'pffmtable', 't9.unicharset'

```
mftraining -F font_properties -U output_unicharset -O t9.unicharset t9.ZYHei_GB18030_c.exp0.tr
```

This step clusters the features from ‘.tr’ and ‘output_unicharset’ files

### 2.2.7 Clustering 2 - generate 'normproto'

```
cntraining t9.ZYHei_GB18030_c.exp0.tr
```

The ‘cntraining’ clusters the features from ‘.tr’ file alone.

### 2.2.8 rename ‘pffmtable’ ‘shapetable’ ‘inttemp’ ‘normproto’ ‘unicharset’

E.g. ‘pffmtable’ --> ‘t9.pffmtable’

### 2.2.9 combine all data with file_head ‘t9.’

```
combine_tessdata t9.
```

### 2.2.10 Copy the ‘traineddata’ file to /usr/share/tesseract-ocr/tessdata

```
sudo cp t9.traineddata /usr/share/tesseract-ocr/tessdata/
```

The generated ‘traineddata’ file should be added to the path, where Tesseract-Engine visits and calls the accordingly traineddata, e.g. the original traineddata file for simplified Chinese ‘chi_sim.traineddata’, for English ‘eng_traineddata’, for German ‘deu.traineddata’.