Training Tesseract 3.04

- Tesseract is an optical character recognition engine for various operating systems.
- Tesseract 3.x is based on traditional computer vision algorithms.
- Version 4 of Tesseract also has the legacy OCR engine of Tesseract 3

Requirements for text input files

Text input files need to meet these criteria:

- ASCII or UTF-8 encoding without BOM
- Unix end-of-line marker (’\n’)
- The last character must be an end of line marker (’\n’).

Python Code for Generating text file

```
import random
import string
random.seed(0)
state_code = ['AP','AR','AS','BR','CG','GA','GJ','HR','HP'
,'JK','JH','KA','KL',
            'MP','MH','MN','ML','MZ','NL','OD','PB','RJ','SK
```

```
','TN','TS','TR','UK',
          'UP','WB','AN','CH','DN','DD','DL','LD','PY']
punc=['-','.',' ']
def randomStringL(stringLength):
    return ''.join(random.choice(state_code))
def randomStringL2(stringLength):
    letters = string.uppercase
    return ''.join(random.choice(letters) for i in range(2
))
def randomStringD(stringLength):
    letters = string.digits
    return ''.join(random.choice(letters) for i in range(s
tringLength))
def randomBet():
    return ''.join(random.choice(punc))
output= ''
for i in range(100):
  rnd = randomBet()
  output = output + (randomStringL(2)+rnd+randomStringD(2)
+rnd+randomStringL2(2)+rnd+randomStringD(4) )+'\n'
#print output

file = open("data.txt","w")
file.write(output)
file.close()
```

Now training_text.txt file is generated.

Training Procedure

1. Generate Training Images and Box Files

1.1 Automated Method

- Run the following command for each font in turn to create a matching tif/box file pair.
- input: UTF-8 text file (training_text.txt) containing our training text.

```
$ text2image --text=training_text.txt --outputbase=eng.FreeSerifBold.exp0 --font='FreeSerif Bold' --fonts_dir=/usr/share/fonts
```

- Three files (eng.FreeSerifBold.exp0.box, eng.FreeSerifBold.exp0.tif, lang.unicharset) created in this step.

2. Run Tesseract for Training

For each of our training image, boxfile pairs, run Tesseract in training mode:

```
$ tesseract eng.FreeSerifBold.exp0.tif eng.FreeSerifBold.exp0 box.train
```

- The output of this step is tr file which contains the features of

each character of the training page.

3. Generate the unicharset file

- Tesseract’s unicharset file contains information on each symbol the Tesseract OCR engine is trained to recognize.
- Currently, generating the unicharset file is done in two steps using these commands: unicharset_extractor and set_unicharset_properties.

3.1 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:

```
$ unicharset_extractor eng.FreeSerifBold.exp0.box
```

- will create unicharset file.

3.2 set_unicharset_properties

- This step allow the addition of extra properties in the unicharset.

```
$ set_unicharset_properties --script_dir=/home/demo/Shishpal/project_tess/tesseract/langdata -U unicharset -O output_unicharset
```

- will create output_unicharset file.

4. The font_properties file

- 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
- Use default font_properties file

5. Clustering

- When the character features of all the training pages have been extracted, we need to cluster them to create the prototypes.
- The character shape features can be clustered using the shapeclustering, mftraining and cntraining programs:

5.1 mftraining

- mftraining will output two other data files: inttemp (the shape prototypes) and pffmtable (the number of expected features for each character).
- mftraining will produce a shapetable file because we didn't run shapeclustering.

```
$ mftraining -F font_properties -U unicharset -O lang.unic
harset eng.FreeSerifBold.exp0.tr
```

5.2 cntraining

- This will output the normproto data file (the character normalization sensitivity prototypes).

```
$ cntraining eng.FreeSerifBold.exp0.tr
```

6. Putting it all together

- now collect together all the files (shapetable, normproto, inttemp, pffmtable, unicharset) and rename them with a lang. prefix (for example eng.)
- run combine_tessdata on them as follows:

```
$ combine_tessdata eng.
```

- The resulting eng.traineddata goes in our tessdata directory.
- Tesseract can now recognize text in our language (in theory) with the following:

```
tesseract image.png output -l eng
```