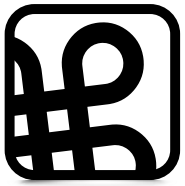


Macoun

Optical Character Recognition auf dem iPhone

Hendrik von Prince

Ablauf

- Eckdaten von Tesseract
- Funktionsweise von Tesseract
- Einbindung in einer iOS-App
- Ergebnisse verbessern
- Ausblick

Eckdaten von Tesseract

1. 1984 bei HP entstanden
2. Seit 2005 von Google weiterentwickelt
3. Stabile Version 3.05.02 vom 19. Juni 2018
4. Apache Lizenz
5. De-Facto-Standard zum Erkennen von Text

Funktionsweise von Tesseract

1. Initialisiere Tesseract mit einem trainierten Modell
2. Bild-Vorverarbeitung
3. Seiten-Layout-Analyse
4. Schriftlinien- und Wort-Detektion
5. Worterkennung (erste Phase)
6. Worterkennung (zweite Phase)

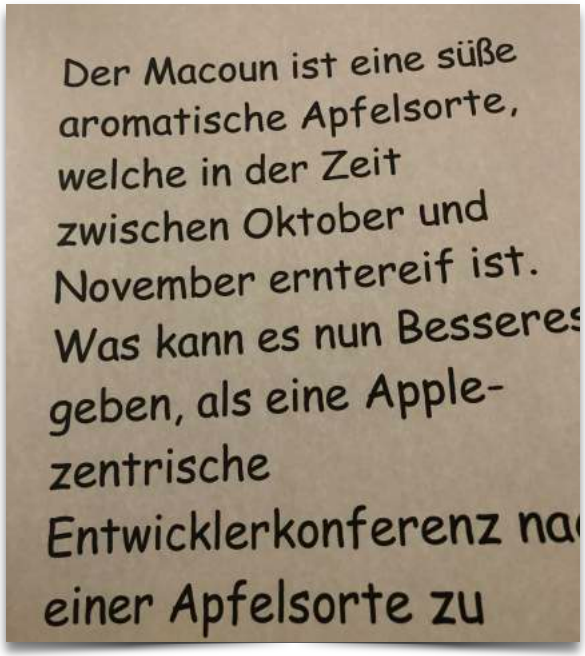
Funktionsweise von Tesseract

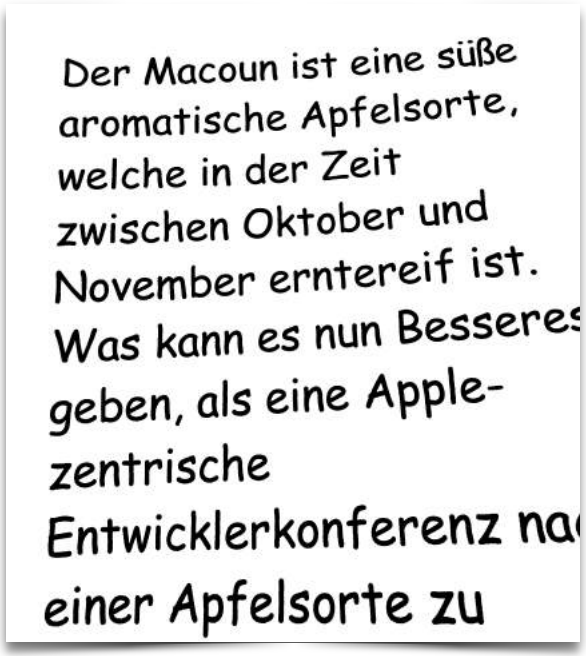
INITIALISIERE TESSERACT MIT EINEM TRAINIERTEN MODELL

- Ein Modell ist nur für eine Sprache trainiert
- Kann aber gleichzeitig für mehrere Fonts trainiert worden sein

1. Bild Vorverarbeitung

UMWANDLUNG IN SCHWARZ-WEIß

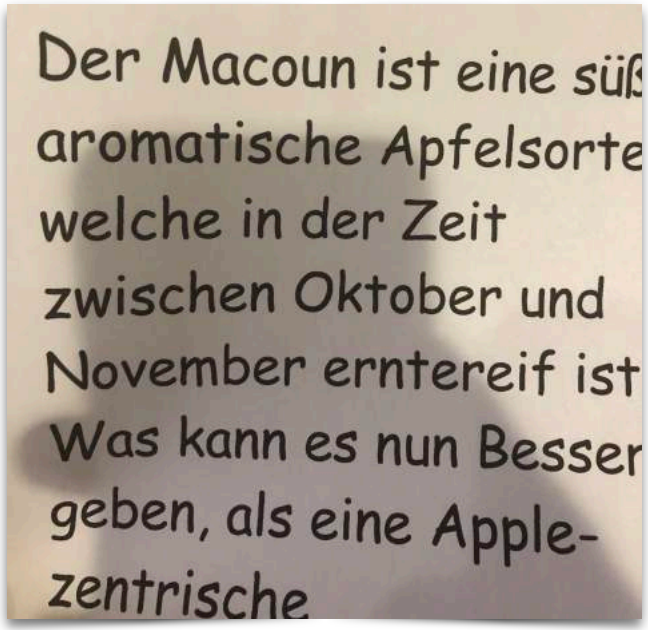


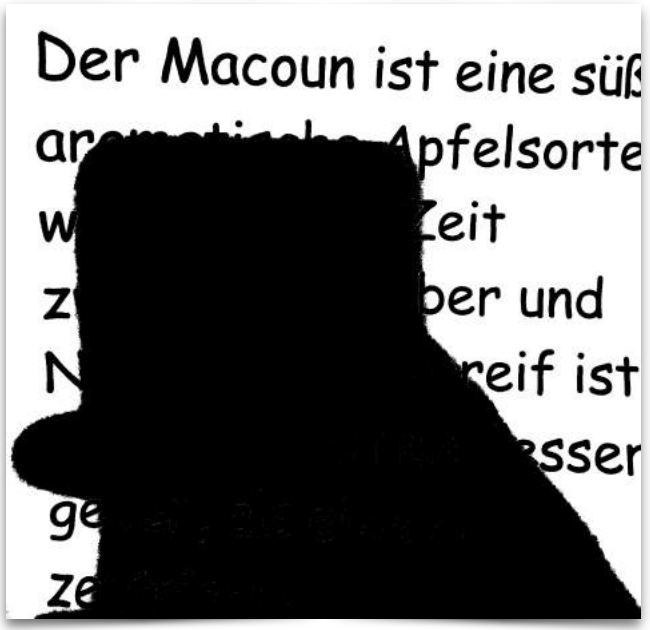


Zitiert von: https://macoun.de/about

1. Bild Vorverarbeitung

STOLPERFALLE: UMWANDLUNG IN SCHWARZ-WEIß

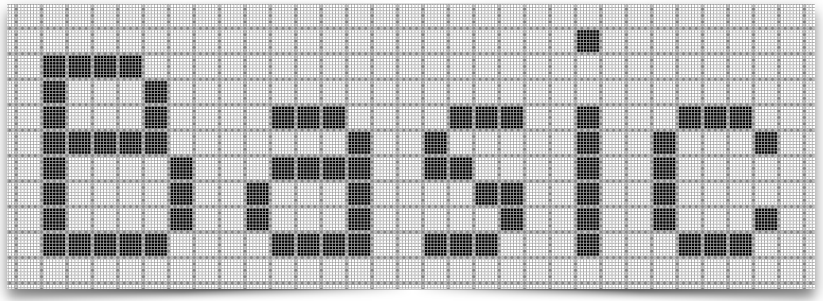




Zitiert von: https://macoun.de/about

2. Seiten-Layout-Analyse

- Erkenne zusammenhängede Pixel-Regionen (Connected Component Analysis)



Bildauthor: Vijay.r.nadar (https://commons.wikimedia.org/wiki/File:OCRBasic.png)

- Unterscheide Text von Nicht-Text (Hartkodierte, heuristikbasierte Logik)

Schriftlinien- und Wort-Detektion

- Erkenne Zeilen
- Finde die Schriftlinie pro Zeile

Bildquelle: An Overview of the Tesseract OCR Engine (2007), Seite 2

Schriftlinien- und Wort-Detektion

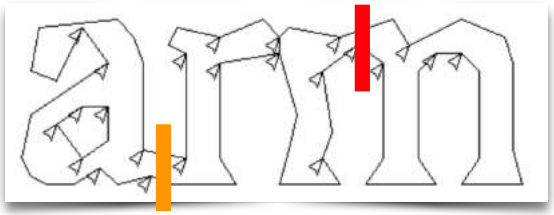
- Für jede Zeile:
  - Unterscheidung von Monospace / Nicht-Monospace-Fonts
  - Gruppiere zusammenhängende Pixel-Regionen zu Wortregionen

Worterkennung (erste Phase)

- Für Monospace-Fonts:
  - Unterteile jede Wortregion in gleichgroße Teile
- Für Nicht-Monospace-Fonts:
  - Unterteile jede Wortregion anhand der Zwischenräume der einzelnen enthaltenen Pixelregionen
- Klassifizierung der enthaltenen Pixelregionen als Zeichen
- Eintragen aller erkannten Worte in ein internes Wörterbuch

Worterkennung (zweite Phase)

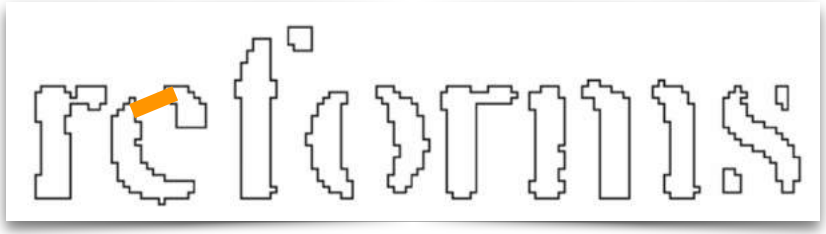
- Durchlaufe alle erkannten Worte und bewerte sie anhand von einem vorgegebenen Wörterbuch und dem internen Wörterbuch
- Solange die Bewertung vom Wort zu ungeau ist, werden die Zeichen mit der geringsten Genauigkeit weiter unterteilt



Bildquelle: An Overview of the Tesseract OCR Engine (2007), Seite 3

Worterkennung (zweite Phase)

- Anschließend werden Pixelregionen zusammengeführt



Bildquelle: An Overview of the Tesseract OCR Engine (2007), Seite 3

- Bewertung vom resultierenden Wort anhand von den Wörterbuchern
- Wähle das Wort mit der beste Bewertung

Zusammenfassung

RESISTENT GEGEN

- Schlechte Bildqualität
- Rotierten Texte
- Fontvariationen
- Sich wiederholenden fachspezifischen Wörtern
- Schwarz/weiß weiß/schwarz

Zusammenfassung

SCHWIERIGKEITEN ERGEBEN SICH MIT

- Dreckigem / gemustertem Hintergrund
- Starken halbseitigen Schatten
- Unterschiedlichen Fonts / Fontgrößen innerhalb einer Zeile
- Wechselnde Fontfarben
- Vom Training stark abweichende Fonts

Einbindung in einer iOS-App

COCOAPOD TESSERACTOCRIOS

- Basiert auf Tesseract 3.03-rc1
- Wird kaum weiterentwickelt, funktioniert aber noch
- Sehr gute API
- Keine Bitcode-Unterstützung

Einbindung in einer iOS-App

COCOAPOD TESSERACT-OCR-IOS

```
inhibit_all_warnings!

target 'TesseractDemo' do
  use_frameworks!

  pod 'TesseractOCRiOS'
end
```

Einbindung in einer iOS-App

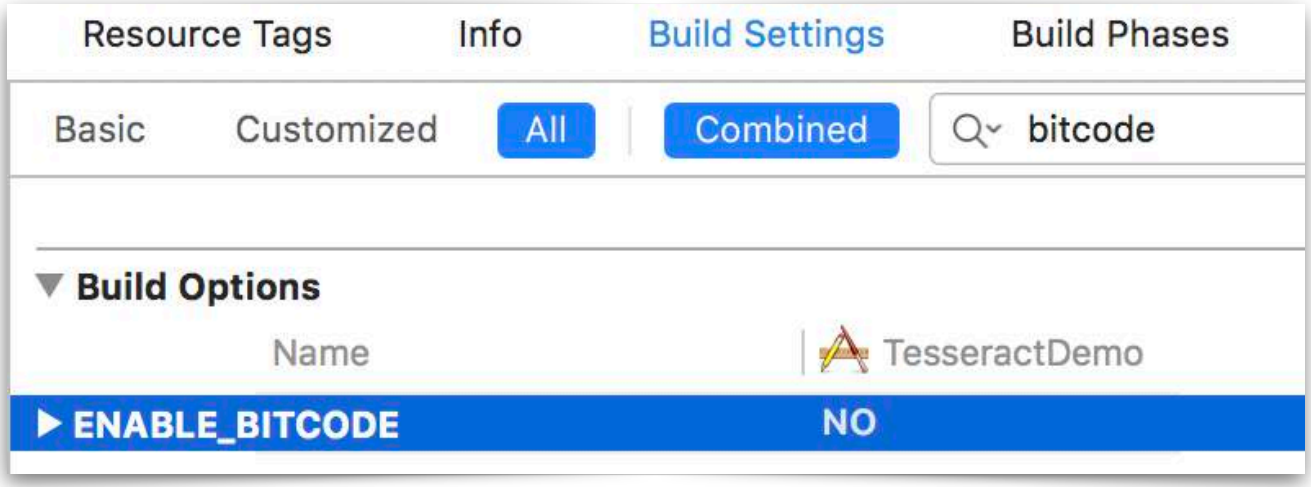
COCOAPOD TESSERACT-OCR-IOS

```
post_install do |installer|
    installer.pods_project.targets.each do |target|
        target.build_configurations.each do |config|
            config.build_settings['ENABLE_BITCODE'] = 'NO'
        end
    end
end
```

Einbindung in einer iOS-App

COCOAPOD TESSERACT-OCR-IOS

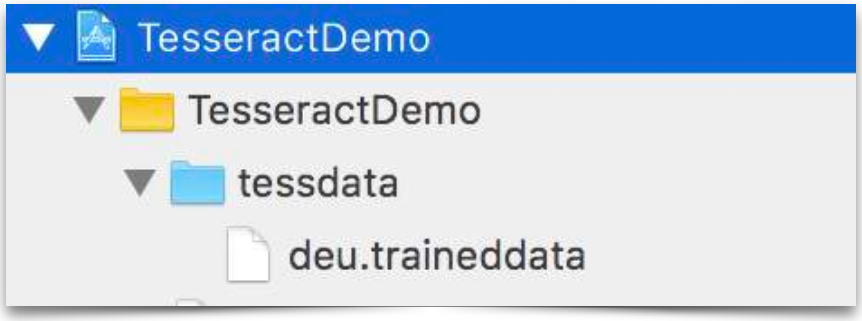
- Bitcode im App-Target deaktivieren



Einbindung in einer iOS-App

SPRACHDATEIEN EINBINDEN

- *tessdata*-Verzeichnis anlegen und als *Folder Reference* in das Projekt ziehen
- Trainierte Sprachdateien von https://github.com/tesseract-ocr/tessdata/tree/3.04.00 herunterladen und in das *tessdata*-Verzeichnis ablegen



Einbindung in einer iOS-App

TESSERACT AUF EIN BILD ANWENDEN

```
import TesseractOCR

let tesseract = G8Tesseract(language: “deu")!
tesseract.image = UIImage(named: "testbild")
tesseract.recognize()
print(tesseract.recognizedText)
```

Einbindung in einer iOS-App

MEHRERE SPRACHEN LADEN

```
let tesseract = G8Tesseract(language: "deu+eng")
```

Ergebnisse verbessern

- Tesseracts Bildvorverarbeitung austauschen
- Tesseract-Parameter tunen
- Vision
- Eigene Fonts antrainieren

Tesseracts Bildverarbeitung austauschen

- Implementieren von `G8TesseractDelegate`
- Filter aus dem Pod `GPUImage` verwenden
- Eigene Bildverarbeitungsalgorithmen in `CIFilter` implementieren und auf den Use-Case anpassen

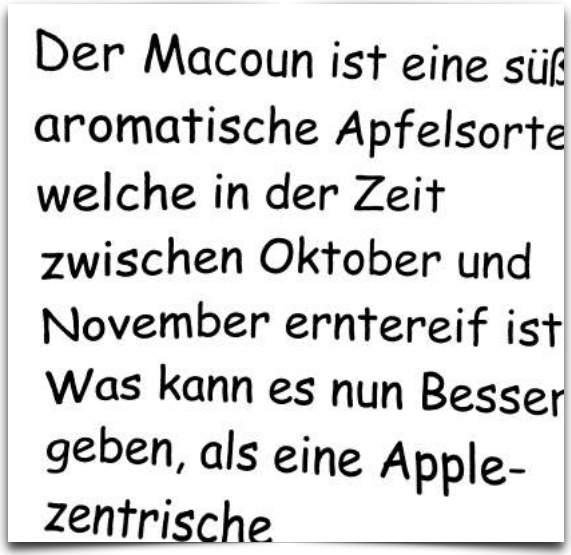
Tesseracts Bildverarbeitung austauschen

G8TESSERACTDELEGATE + GPUIMAGE

```
func preprocessedImage(for tesseract: G8Tesseract!,
                       sourceImage: UIImage!) -> UIImage!
{
  let filter = GPUImageLuminanceThresholdFilter()
  filter = 0.3
  return filter.image(byFilteringImage: sourceImage)
}
```

Tesseracts Bildverarbeitung austauschen

`G8TESSERACTDELEGATE + GPUIMAGE`




```
Der‘ Macoun ist eine süß
aromatische Apfdsorfe welche
in der Zeit zwischen Okfober
und November‘ ernfereif isT
Was kann es nun Besser
geben, als eine Apple-
zentrische
```

Tesseract-Parameter tunen

```
> tesseract --print-parameters
```

Tesseract-Parameter tunen

```
> tesseract --print-parameters
Tesseract parameters:
editor_image_xpos 590 Editor image X Pos
editor_image_ypos 10 Editor image Y Pos
editor_image_menuheight 50 Add to image height for menu bar
editor_image_word_bb_color 7 Word bounding box colour
editor_image_blob_bb_color 4 Blob bounding box colour
editor_image_text_color 2 Correct text colour
editor_dbwin_xpos 50 Editor debug window X Pos
editor_dbwin_ypos 500 Editor debug window Y Pos
editor_dbwin_height 24 Editor debug window height
editor_dbwin_width 80 Editor debug window width
```

Tesseract-Parameter tunen

```
> tesseract --print-parameters
editor_word_xpos 60  Word window X Pos
editor_word_ypos 510 Word window Y Pos
editor_word_height 240 Word window height
editor_word_width  655 Word window width
textord_debug_tabfind 0 Debug tab finding
textord_debug_bugs 0 Turn on output related to bugs in tab
finding
textord_testregion_left -1  Left edge of debug reporting rectangle
textord_testregion_top  -1  Top edge of debug reporting rectangle
textord_testregion_right 2147483647  Right edge of debug
rectangle
```

Tesseract-Parameter tunen

```
> tesseract --print-parameters
textord_testregion_bottom  2147483647  Bottom edge of debug
rectangle
textord_tabfind_show_partitions 0 Show partition bounds, waiting
if >1
devanagari_split_debuglevel  0 Debug level for split shiro-rekha
process.
edges_max_children_per_outline  10  Max number of children inside
a character outline
edges_max_children_layers  5 Max layers of nested children inside
a character outline
edges_children_per_grandchild 10  Importance ratio for chucking
```

Tesseract-Parameter tunen

```
> tesseract --print-parameters
textord_test_y -2147483647 coord of test pt
textord_min_blobs_in_row 4 Min blobs before gradient counted
textord_spline_minblobs 8 Min blobs in each spline segment
textord_spline_medianwin 6 Size of window for spline segmentation
textord_max_blob_overlaps  4 Max number of blobs a big blob can
overlap
textord_min_xheight 10  Min credible pixel xheight
textord_lms_line_trials 12  Number of linew fits to do
oldbl_holed_losscount 10  Max lost before fallback line used
pitsync_linear_version  6 Use new fast algorithm
pitsync_fake_depth 1 Max advance fake generation
```

Tesseract-Parameter tunen

```
+ 650 weitere
```

Tesseract-Parameter tunen

```
let minXHeight = "\(Int(image.size.height * 0.05))"
tesseract.setVariableValue(minXHeight, forKey: "textord_min_xheight")
tesseract.setVariableValue("‘", forKey: "tessedit_char_blacklist")
```

```
Der Macoun ist eine süß aromatische
Apfelsorfe welche in der Zeit zwischen
Okfober und November ernfereif isT Was
kann es nun Besser geben, als eine
Apple- zentrische
```

Tesseract-Parameter tunen

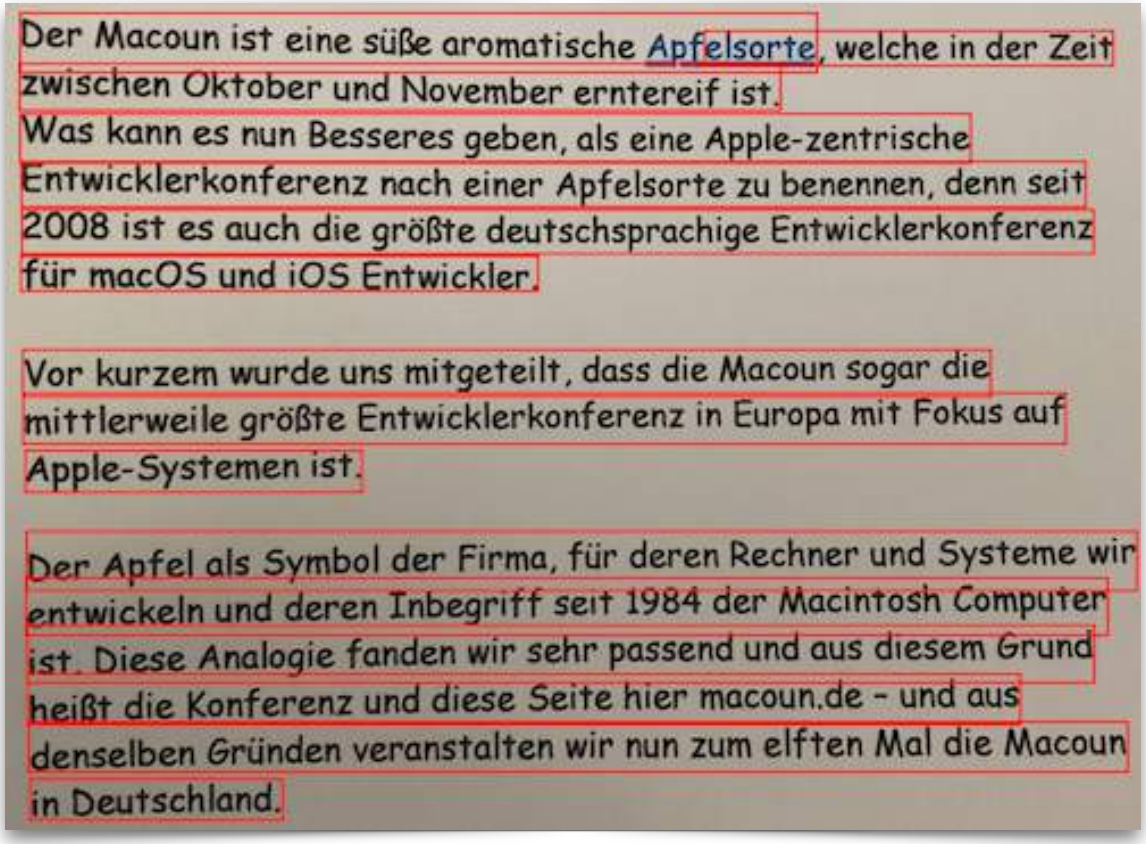
- `tessedit_char_blacklist / tessedit_char_whitelist`
- `textord_noise_hfract, textord_heavy_nr`
- `load_system_dawg, load_number_dawg, load_fixed_length_dawgs`
- `load_bigram_dawg`
- Kurze Beschreibung zu den einzelnen Parametern: http://www.sk-spell.sk.cx/tesseract-ocr-parameters-in-302-version

Vision

- Framework von Apple ab iOS 11.0
- Detektiert Rechtecke, Gesichter, Barcodes,… und Text
- Texterkennung funktioniert unabhängig von der Sprache und Font
- Sehr performant

  => Berechne mit Vision in welchem Bildbereich sich Text befindet und übergebe nur diesen an Tesseract

Vision



Der‘ Macoun ist eine süße aromatische Apfelsorte, welche in der Zeit
zwischen Oktober und November erntereif ist.
[…]
Diese Analogie fanden wir sehr passend und aus diesem Grund heißt die Konferenz und diese Seite hier macoun.de - und aus denselben Gründen veranstalten wir nun zum elften Mal die Macoun in Deutschland.

Vision

```
import Vision

let handler = VNImageRequestHandler(ciImage: image,
                                    orientation: .up,
                                    options: [:])
let textDetect = VNDetectTextRectanglesRequest()
let requests = [textDetect]
try? handler.perform(requests)
```

Vision

```
let results = textDetect.results as! [VNTextObservation]
let boundingBoxes = results.map({ $0.boundingBox })
var regionWithText = combine(boundingBoxes)
regionWithText = regionWithText.insetBy(dx: -50, dy: -50)

Let croppedCGImage = image.cgImage!.cropping(to: regionWithText)!

tesseract.image = UIImage(cgImage: croppedCGImage)
```

Vision

DC!‘ Macoun ist eine süße aromatische Apfelsarte, welche in der Zeit
zmschen Oktober und November erntereif ist.
[…]
und aus
  Gründen veranstalten wir nun zum elften Mal die Macoun
 \' „ \' \' r „ - “ . ‘ \\

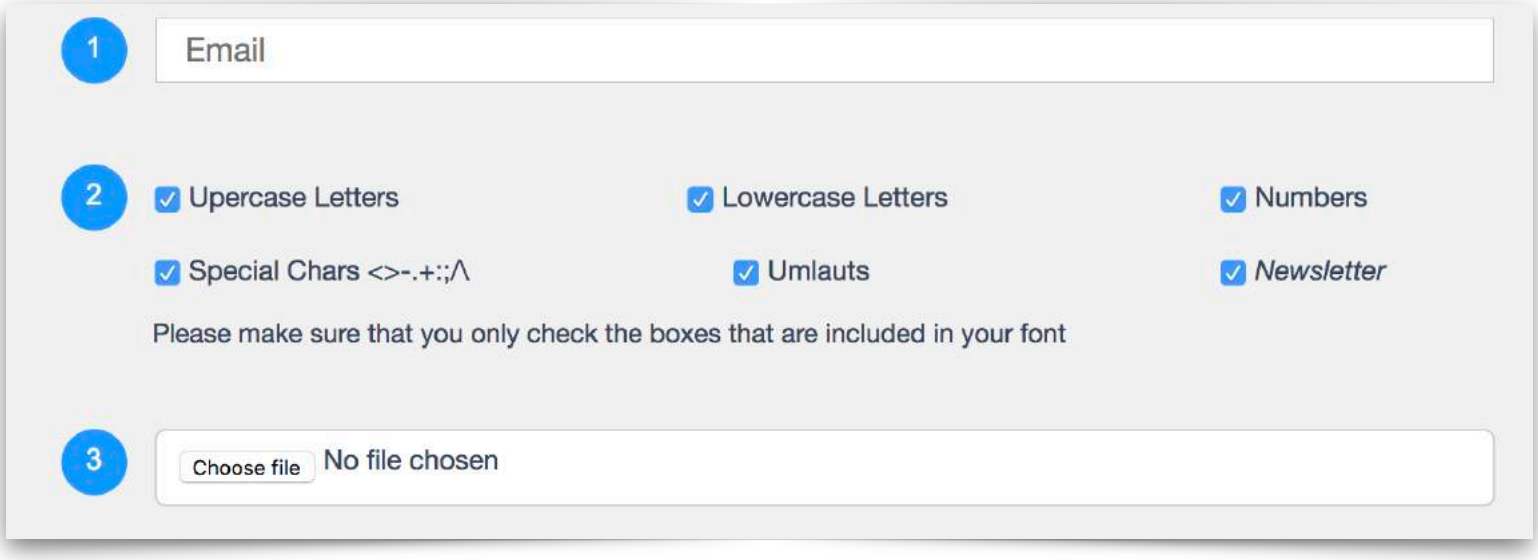
61 Fehler

Der‘ Macoun ist eine süße aromatische
Apfelsorte, welche in der Zeit
zwischen Oktober und November erntereif ist.
[…]
und aus
denselben Gründen
veranstalten wir nun zum elften Mal die Macoun
in Deutschland.

1 Fehler

Eigene Fonts antrainieren

- Variante eins: http://trainyourtesseract.com/



Bildquelle: http://trainyourtesseract.com/, Abgerufen am 12.09.2018

Eigene Fonts antrainieren

HTTP://TRAINYOURTESSERACT.COM/

- Liefert kleine Dateien mit Ergebnissen die für manche Anwendungsfälle ausreichen können

Eigene Fonts antrainieren

- Variante 2: Per Hand trainieren

Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Tesseract 3.05 über homebrew installieren
- Trainings-Scripte installieren
- Trainings-Script für mac OS patchen
- Sprachdaten herunterladen
- Training starten

Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Tesseract 3.05 über homebrew installieren

```
brew install tesseract --with-all-languages --with-training-tools
```

Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Trainings-Scripte installieren

```
curl https://raw.githubusercontent.com/tesseract-ocr/tesseract/3.05/training/
tesstrain.sh -o /usr/local/Cellar/tesseract/3.05.02/bin/tesstrain.sh
curl https://raw.githubusercontent.com/tesseract-ocr/tesseract/3.05/training/
language-specific.sh -o /usr/local/Cellar/tesseract/3.05.02/bin/language-
specific.sh
curl https://raw.githubusercontent.com/tesseract-ocr/tesseract/3.05/training/
tesstrain_utils.sh -o /usr/local/Cellar/tesseract/3.05.02/bin/tesstrain_utils.sh
chmod +x /usr/local/Cellar/tesseract/3.05.02/bin/tesstrain.sh
```

Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Trainings-Script für mac OS patchen

```
printf 'diff --git a/tesstrain_utils.sh b/tesstrain_utils.sh
                index 906a20ac..0701b80a 100755
                --- a/tesstrain_utils.sh
                +++ b/tesstrain_utils.sh
                @@ -28,2 +28,3 @@ EXTRACT_FONT_PROPERTIES=1
                 WORKSPACE_DIR=`mktemp -d`
                +export PANGOCAIRO_BACKEND=fc
```

Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Sprachdaten herunterladen

```
git clone https://github.com/tesseract-ocr/langdata
```

Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Training starten

```
/usr/local/Cellar/tesseract/3.05.02/bin/tesstrain.sh --lang deu --langdata_dir ./
langdata/ --tessdata_dir /usr/local/Cellar/tesseract/3.05.02/share/tessdata --
fontlist "Comic Sans MS" --fonts_dir ~/custom-fonts
```

- “deu”: Der Font wird mit den Parametern für die deutsche Sprache trainiert
- “Comic Sans MS”: Der zu trainierende Font
- --fonts_dir: Nur nötig falls der Font nicht im System-Ordner liegt

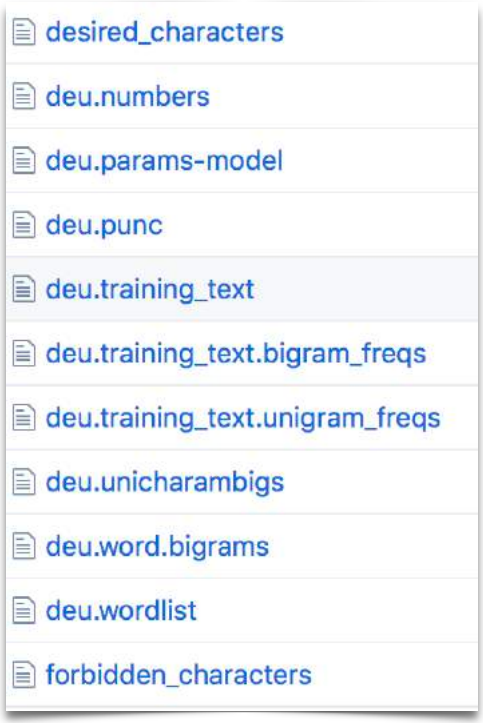
Eigene Fonts antrainieren

PER HAND TRAINIEREN

- Am Ende liegt in `/tmp/tesstrain/tessdata` die Datei `deu.traineddata`
- Ca. 12MB groß
- Training basiert auf mehreren Dateien die in `langdata/deu` enthalten sind
- Optimierbar für andere Fonts, Trainingstext, Characterähnlichkeiten, …

Eigene Fonts antrainieren

PER HAND TRAINIEREN



Zusammenfassung

- Tesseract initial einzusetzen ist sehr einfach
- Vermutlich sind die Ergebnisse anfangs relativ schlecht
- Mit den gezeigten Vorgehensweisen können sie signifikant verbessert werden
- 100% Genauigkeit: Stellt euch auf mehrere Wochen rumprobieren und Schmerzen ein.

Ausblick

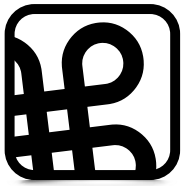
- Tesseract 4.0 mit Neuronalem Netz
  - Erste Alpha im November 2016
  - Aktuelle Beta vom 2. August
  - Pull-Request für TesseractOCRiOS
  - Sehr viel bessere Ergebnisse, dafür sehr viel langsamer

Ausblick

- Moderne “Alternative” SwiftOCR?
  - Laut eigenen Angaben nur für kurze (einzeilige) Texte geeignet, dann aber sehr viel schneller und genauer als Tesseract
- API von Apple?
  - Das Know-How besitzen sie
  - Allerdings weißt bisher nichts darauf hin

Fragen?

Vielen Dank



Macoun