

# 1 Implementierung

Im Allgemeinen war bereits ein Großteil des benötigten Quellcodes mit der Angabe gegeben. Der gegebene Quelltext wurde um einige Methoden und Funktionen erweitert um die benötigte Funktion abbilden zu können. Bei der Implementierung wurde laut Angabe vorgegangen.

Zusätzlich wird der Code um automatisierte Tests sowie Funktionen zur visuellen Kontrolle ergänzt. Zur Überprüfung der Erkennung kann ein Overlay erzeugt werden, das die detektierten Zeichenbereiche im Originalbild hervorhebt. Debugging-Funktionen sind implementiert, jedoch im finalen Code standardmäßig auskommentiert.

## 1.1 Zerteilen des Bildes in Teilbilder je Zeichen

Das Eingabebild wird zunächst in Graustufen eingelesen und binarisiert. Anschließend erfolgt eine zweistufige Segmentierung der Zeichen:

1. **Horizontale Segmentierung:** Es werden Bildzeilen identifiziert, in denen Zeichen vorhanden sind. Dabei wird geprüft, ob eine Bildreihe ausschließlich Hintergrund enthält.
2. **Vertikale Segmentierung:** In den gefundenen Zeilen werden die einzelnen Zeichen durch Analyse der Spalten separiert. Auch hier wird überprüft, ob eine Spalte nur Hintergrund enthält.

Für jedes segmentierte Zeichen wird ein `SubImageRegion`-Objekt erzeugt, welches die Position, Höhe und Breite des Bereichs im Originalbild beschreibt. Zusätzlich wird eine Methode zur minimalen Höhenanpassung (`minimize_character_bounding_height`) angewendet, um den Zeichenbereich vertikal anzupassen.

## 1.2 Klassifizierung

Um die Teilbilder der einzelnen Zeichen einem konkreten Zeichen zuzuordnen, werden Features verwendet. Diese Features sind Merkmale, anhand welcher Zeichen identifiziert werden können. Die Kombination dieser Features soll in Folge möglichst alle Zeichen eindeutig klassifizieren. Um alle Features im Ergebnis gleich zu gewichten, müssen diese normalisiert werden. Diese wurde bereits in der Angabe mitgeliefert. Folgende Features sind implementiert:

- **FGcount:** Zählt die Anzahl der Vordergrundpixel (also die Zeichenpixel) innerhalb eines Zeichenbereichs.
- **MaxDistX:** Absolute Breite der Bounding-Box
- **MaxDistY:** Absolute Höhe der Bounding-Box
- **AspectRatio:** Relatives Verhältnis zwischen Breite und Höhe
- **FgBgRatio:** Relatives Verhältnis zwischen Vordergrund und Hintergrund
- **VerticalAsym:** Symetrie auf der vertikalen Achse (Es wird eine lineare Funktion vom linken unteren Pixel zum rechten oberen erzeugt  $\Rightarrow f(x) = kx + d$ , wobei  $d$  immer 0 ist, und jeder Pixel mit diesem Gewichtungsfaktor aufsummiert)

- **HorizontalAsym:** Symetrie auf der horizontalen Achse (Es wird eine lineare Funktion vom linken oberen Pixel zum rechten unteren erzeugt  $\Rightarrow f(x) = kx + d$ , wobei  $d$  immer 0 ist, und jeder Pixel mit diesem Gewichtungsfaktor aufsummiert)

## 2 Test

Um tests einfach wiederholbar zu gestalten sind diese in Form von statisch definierten Testfällen implementiert. Inhalt dieser Testfälle sind sämtliche Vorkommnisse aller Zeichen im Beispiel-Bild. Die Testfälle prüfen dabei immer, ob das erwartete Vorkommen eines Zeichens auch tatsächlich gefunden wurde. Um eine solche Testmöglichkeit bereitzustellen ist die gegebene runMethode entsprechend erweitert:

- Es wird die Anzahl an gefunden Zeichen als Ergebnis zurückgegeben
- Die Schnittstelle bietet die Möglichkeit das Referenzzeichen von extern parameterieren zum können

```
1 def run(self, img_path, tgtCharRow: int, tgtCharCol: int) -> int:
2     ...
```

language=Python

Die Testfälle an sich können für das gegebene Testbild direkt via der main Methode ausgeführt werden. Zu diesen „automatisierten“ Tests wurden zusätzlich noch manuelle Tests durchgeführt um das Splitten, Minimieren und Einfärben der Bounding-Boxes zu testen. Diese Tests sind manuell durchzuführen und optisch zu prüfen.

### 2.1 Ergebnis

#### 2.1.1 Automatisierte Tests


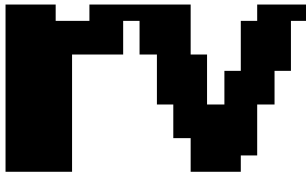


```
1  SUCCESS character "e" - expected: 169 found: 169
2  SUCCESS character "n" - expected: 115 found: 115
3  SUCCESS character "s" - expected: 102 found: 102
4  SUCCESS character "a" - expected: 92 found: 92
5  FAILED character "t" - expected: 82 found: 78
6  FAILED character "r" - expected: 81 found: 75
7  SUCCESS character "d" - expected: 69 found: 69
8  SUCCESS character "i" - expected: 57 found: 57
9  SUCCESS character "h" - expected: 45 found: 45
10 SUCCESS character "u" - expected: 39 found: 39
11 SUCCESS character "o" - expected: 38 found: 38
12 SUCCESS character "m" - expected: 37 found: 37
13 SUCCESS character "l" - expected: 35 found: 35
14 SUCCESS character "c" - expected: 33 found: 33
15 SUCCESS character "g" - expected: 27 found: 27
16 FAILED character "w" - expected: 25 found: 24
17 SUCCESS character "G" - expected: 23 found: 23
18 SUCCESS character "b" - expected: 22 found: 22
19 SUCCESS character "." - expected: 20 found: 20
20 SUCCESS character "," - expected: 16 found: 16
21 SUCCESS character "L" - expected: 10 found: 10
22 FAILED character "v" - expected: 10 found: 9
23 SUCCESS character "E" - expected: 8 found: 8
24 SUCCESS character "W" - expected: 8 found: 8
25 SUCCESS character ":" - expected: 8 found: 8
26 SUCCESS character "T" - expected: 8 found: 8
27 SUCCESS character "D" - expected: 8 found: 8
```

```

28 SUCCESS character "S" - expected: 8 found: 8
29 SUCCESS character "A" - expected: 7 found: 7
30 SUCCESS character "ü" - expected: 7 found: 7
31 SUCCESS character "ö" - expected: 7 found: 7
32 FAILED character "f" - expected: 6 found: 5
33 SUCCESS character "F" - expected: 6 found: 6
34 FAILED character "z" - expected: 6 found: 5
35 SUCCESS character "H" - expected: 5 found: 5
36 SUCCESS character "P" - expected: 5 found: 5
37 SUCCESS character "M" - expected: 4 found: 4
38 SUCCESS character "B" - expected: 3 found: 3
39 SUCCESS character ";" - expected: 2 found: 2
40 SUCCESS character "U" - expected: 2 found: 2
41 SUCCESS character "N" - expected: 2 found: 2
42 SUCCESS character "k" - expected: 2 found: 2
43 SUCCESS character "j" - expected: 2 found: 2
44 SUCCESS character "P" - expected: 2 found: 2
45 SUCCESS character "ä" - expected: 2 found: 2
46 SUCCESS character "I" - expected: 1 found: 1
47 SUCCESS character "O" - expected: 1 found: 1
48 SUCCESS character "Z" - expected: 1 found: 1
49 SUCCESS character "J" - expected: 1 found: 1
50 =====
51 Tests passed: SUCCESSFULLY 43 / FAILED 6
52 =====

```

Die Tests mit dem gesamten Zeichensatz des Testbilds zeigten eine hohe Erkennungsgenauigkeit. Insgesamt wurden 49 Zeichenklassen getestet, wobei 43 erfolgreich und 6 fehlerhaft erkannt wurden. Diese Problemfälle sind zurückzuführen auf die „primitive“ splitting Methode(Fire-Through-Methode), welche für das Beispiel-Bild nicht immer alle Zeichen richtig zerteilt. Es gibt Zeichenfolgen, die nicht zum Teil überlappend sind und somit nicht richtig getrennt werden. Diese Fehler werden in der folgenden Tabelle genauer gezeigt.

rt	 <p>Abbildung 1: Fehlerhafte Segmentierung: <math>r</math> und <math>t</math> werden als ein Zeichen erkannt</p>
rv	 <p>Abbildung 2: Fehlerhafte Segmentierung: <math>r</math> und <math>v</math></p>
rf	 <p>Abbildung 3: Fehlerhafte Segmentierung: <math>r</math> und <math>f</math></p>
zw	 <p>Abbildung 4: Fehlerhafte Segmentierung: <math>z</math> und <math>w</math> verschmelzen zu einem Zeichen</p>

In Summe sind davon alle Zeichen betroffen, welche in der Auswertung fehlgeschlagen sind.

### 2.1.2 Beispiel: Test von erkannten e

In diesem Beispiel ist demonstriert wie beispielsweise das Zeichen „e“ erkannt wird. Die identifizierten Zeichen werden hier in Graustufe hinterlegt.

**Im Anfang schuf Gott Himmel und Erde; die Erde aber war wüst und wirr, Finsternis lag über der Urflut und Gottes Geist schwebte über dem Wasser. Gott sprach: Es werde Licht. Und es wurde Licht. Gott sah, dass das Licht gut war. Gott schied das Licht von der Finsternis und Gott nannte das Licht Tag und die Finsternis nannte er Nacht. Es wurde Abend und es wurde Morgen: erster Tag. Dann sprach Gott: Ein Gewölbe entstehe mitten im Wasser und scheide Wasser von Wasser. Gott machte also das Gewölbe und schied das Wasser unterhalb des Gewölbes vom Wasser oberhalb des Gewölbes. So geschah es und Gott nannte das Gewölbe Himmel. Es wurde Abend und es wurde Morgen: zweiter Tag. Dann sprach Gott: Das Wasser unterhalb des Himmels sammle sich an einem Ort, damit das Trockene sichtbar werde. So geschah es. Das Trockene nannte Gott Land und das angesammelte Wasser nannte er Meer. Gott sah, dass es gut war. Dann sprach Gott: Das Land lasse junges Grün wachsen, alle Arten von Pflanzen, die Samen tragen, und von Bäumen, die auf der Erde Früchte bringen mit ihrem Samen darin. So geschah es. Das Land brachte junges Grün hervor, alle Arten von Pflanzen, die Samen tragen, alle Arten von Bäumen, die Früchte bringen mit ihrem Samen darin. Gott sah, dass es gut war. Es wurde Abend und es wurde Morgen: dritter Tag. Dann sprach Gott: Lichter sollen am Himmelsgewölbe sein, um Tag und Nacht zu scheiden. Sie sollen Zeichen sein und zur Bestimmung von Festzeiten, von Tagen und Jahren dienen; sie sollen Lichter am Himmelsgewölbe sein, die**

Abbildung 5:

### 2.1.3 Beispiel: Test von Bounding-Box Minimierung

In diesem Beispiel wird demonstrativ überprüft, ob das minimieren der Bounding-Boxen entsprechend funktioniert. Als Beispielwort ist mit Absicht das Wort Früchte ausgewählt, da dieses einige Besonderheiten wie z.B.: das Zeichen ü aufweist.



Abbildung 6:

## 3 Fragen

### 3.1 Which letters can be detected in a confident way and which letters lead to problems – and why?

Im allgemeinen funktionieren alle Zeichen mit den implementierten Erkennungs Features für das gegebene Testbild gut. Auch jene Zeichen, bei denen zu Beginn angenommen wurde, dass diese Probleme verursachen würden. Wie zum Beispiel b und d, welche jedoch mit der Asymetrie gut unterschieden werden können. Die Probleme kommen vorrangig vom splitten der Zeichen und nicht vom Klassifizieren derer.

### 3.2 Are all fonts applicable to this kind of OCR strategy – why or why not?

Nein, es können nur Schriftarten verwendet werden, welche klar durch gerade horizontale Linien getrennt werden können. Mono-Space Schriftarten eignen sich zum Beispiel sehr gut, wo hingegen Schriftarten, die eine Handschrift nachahmen nicht funktionieren würden.

Ein Beispiel für eine problematische Schriftart ist in Abbildung 7 zu sehen. Aufgrund der überlappenden Zeichen (z.B. *T* und *e*) funktioniert die *Fire-Through*-Methode nicht, da eine eindeutige vertikale Trennung nicht möglich ist.

# Texterkennung

Abbildung 7: Beispiel einer Schriftart mit überlappenden Zeichen

### 3.3 Does classification accuracy depend on the other characters in the specific line – if that's the case: why and how to overcome this issue?

Bei dem angewandten Vorgehen wird nicht unterschieden, wie oft ein Zeichen vorkommt. Jedoch gibt es bei der gegebenen Schriftart Probleme mit überlappenden Zeichen-Konstellationen. Beispiele solcher Reihenfolgen sind im Kapitel 2 ersichtlich.

Um dieses Problem zu umgehen, könnten adaptive Segmentierungsmethoden und kontextabhängige Klassifikatoren verwendet werden, welche die Umgebung eines Zeichens in die Bewertung einbeziehen.

### 3.4 Evaluate confidence per letter and discuss

Für jedes klassifizierte Zeichen wird ein Konfidenzwert berechnet, der angibt, wie stark die Ähnlichkeit zum Referenzzeichen ist. In den durchgeführten Tests zeigte sich, dass ein Konfidenzwert von etwa 0.99999 für das verwendete Testbild sehr gut funktioniert und ein sehr hohes Maß an Übereinstimmung signalisiert. Dieser Wert könnte jedoch bei anderen Bildern auch zu restriktiv sein, wenn Zeichen durch Rauschen oder Segmentierungsfehler leicht verfälscht wurden.

### **3.5 Ensure that the split characters image region is shrinked to its bounding box. How can that help to improve result quality?**

Das Minimieren der Bounding-Boxes hilft vor allem bei Features, welche sich auf die Höhe dieser beziehen, weil somit einzelne Zeichen möglicherweise schon alleinig auf Basis deren Höhe zugeordnet werden können. Würde keine Minimierung durchgeführt werden, hätten alle Bounding-Boxes einer Zeile die selbe Höhe und dieses Feature wäre wenig aussagekräftig.

### **3.6 Discuss the normalization process – how does character occurrence probability influence the results?**

Je öfter ein Zeichen erkannt wird, umso mehr werden Ausreißer gedämpft (durch Mittelwertbildung – Normalisierung).

### **3.7 Discuss how the classification process itself could be improved. Are there better strategies for feature-based classification?**

Im Allgemeinen hat dieser Ansatz den Charm, dass die Implementierung, einfach verständlich ist und zudem noch eine sehr gute Performance liefert. Jedoch können Machine-Learning Modelle vor allem bei komplexeren Schriftarten/Rahmenbedingungen womöglich bessere Ergebnisse liefern.

### **3.8 How does correlation of some of the features itself influence the achievable classification results (e.g. max-distance-from-centroid will somehow be correlated to the specific width)?**

Die Features verhalten sich in einem Fall der Abhängigkeit zueinander propotional (bei dieser Implementierung). Das bedeutet, dass es keinen Effekt auf das Ergebnis gibt, da die konkrete Gewichtung normalisiert wird.