

## 2. Beispiel der Übung Medizinische Bildverarbeitung

LU 183.630 - 2016SS

30. Mai 2016

Fragen an: Markus Krenn: `markus.krenn@meduniwien.ac.at`,  
Markus Holzer: `markus.holzer@meduniwien.ac.at`

### 1 Abgabemodalitäten

- Eine gemeinsame Abgabe pro Team, inkl Angabe der Teammitglieder.
- Abgabe per email an `markus.holzer@meduniwien.ac.at`
- Deadline für die Abgabe ist der 24. Juni 24h.

Inhalt der Abgabe:

- Abzugeben ist der lauffähige Code als `.zip`, filename: `Abgabe-PF-XX.zip`, mit Gruppennummer statt `XX`.
- Es muss ein `RUN.m` File enthalten sein, das ohne Userinteraktion alle Resultate berechnet und die Figures plottet.
- Der Code muss dokumentiert sein (inkl. Angabe, wo welches Unterbeispiel gerechnet wird).
- Weiters ist ein PDF zu attachen, in dem die einzelnen Punkte ausgearbeitet sind (ca. 5-8 Seiten).
- Die Ausarbeitung sollte knapp und präzise sein, bitte keine Code-Teile oder allgemeine Erklärungen inkludieren.
- Die Abgabe kann in Deutsch oder Englisch ausgearbeitet werden.
- Fragen zu Umfang / Inhalt und Feedback zum bereits erstellten Code/Report in den Übungseinheiten oder per Email.

## 2 Angabe

Im zweiten Beispiel geht es darum, das PCA Model aus dem ersten Beispiel in einem Segmentierungsalgorithmus zu verwenden. Es handelt sich dabei um eine vereinfachte Variante von Particle Filters<sup>1</sup>. Behandelte Themen:

- Feature-Berechnung
- Klassifikation und Feature-Selection mittels Random Forests
- Aufstellung und Optimieren einer Kostenfunktion, d.h. Segmentierung

### 2.1 Vorhandene Daten bzw Hilfsfunktionen mit Dokumentation im Source Code:

- Daten sind in `handdata.mat`. `images` enthält die Bilder, `masks` die Masken mit den Konturen der Objekte, `landmarks` die  $n_{dim} \times n_{landmarks} \times n_{bones} \times n_{images}$  landmarks und `aligned` die bereits alignen Landmarks für jeden Knochen für die PCA.
- `computeHaarLike.m` berechnet Haar-like Features für ein Bild
- `optimize.m` optimiert eine gegebene Kostenfunktion
- `optimizeDEMO.m` demonstriert `optimize.m`
- `cache.m` erlaubt das einfache Zwischenspeichern von Ergebnissen (siehe `cacheDEMO.m`)

Evtl benötigte Matlab Funktionen sind unter anderem: `gradient`, `meshgrid`, `TreeBagger`, `randperm`.

### 2.2 Fragestellung

In Klammer jeweils die erreichbare Punktezahl, insgesamt 40 Punkte. Die Bilder 1-30 sind für das Training (dh PCA Modell und Klassifikator) und Bilder 31-50 für das Testen zu verwenden.

1. **Shape-Modell (5 Punkte)** Erweitern Sie Funktion `generateShape.m` so, dass die Shapes entsprechend den Paramtern  $r, s, x, y$  rotiert, skaliert und verschoben werden können (Stichwort Rotationsmatrix), dh die Funktion hat dann  $n_{modes} + 4$  Parameter:  $p = (b, scaling, rotation, x - translation, y - translation)$ . Plotten Sie analog zum ersten Beispiel Shapes für mehrere Werte von Skalierung und Rotation.

---

<sup>1</sup>siehe PDF `deBruijne2004MICCAI_ParticleFilters.pdf`

2. **Featureberechnung (7 Punkte)** – Schreiben Sie eine Funktion `computeFeatures(image)`, die für ein Bild die folgenden Features berechnet und als  $n_{features} \times n_{pixels}$  Matrix retourniert:

- Grauwert des Pixels
- Gradient in x- und y-Richtung
- Stärke des Gradienten
- Haar-like Features<sup>2</sup>, berechnet auf dem Grauwertbild mithilfe der Funktion `computeHaarLike.m`.
- Haar-like Features, berechnet auf der Gradientenstärke
- x- und y-Koordinaten des Pixels

Stellen Sie diese Features für Bild 1 mit `imagesc` dar (von den Haar-like jeweils nur das erste Feature). Sobald die Featureberechnung funktioniert, kann sie einfach mit `cache` gecacht werden. Gerne können auch weitere Features berechnet und evaluiert werden!

3. **Klassifikation & Feature-Selection (11 Punkte)** Die Features werden nun verwendet, um einen Klassifikator zu trainieren, der die Kanten des zu segmentierenden Objekts klassifizieren soll. Wir verwenden Random Forests<sup>3</sup>, die in Matlab in der Klasse `TreeBagger` implementiert sind.

- (a) Schreiben Sie eine Funktion `train(images, masks)`, die für jedes Bild die Features berechnet und dann für alle Trainingsbilder einen Random Forest trainiert, mit den Masks als Klassenlabels. Ein Beispielaufruf sieht z.B. so aus:

```
rf = TreeBagger(32, features', labels', 'OOBVarImp', 'on');
```

Hinweis: Um das Trainieren zu beschleunigen, verwenden Sie alle Pixel der Knochenkontur, aber nur ein zufällig ausgewähltes Subset der Hintergrundpixel (gleichviele Samples für Kontur/Hintergrund).

- (b) Untersuchen und Interpretieren Sie den Einfluss der Anzahl von Trees mittels `oobError`
- (c) Untersuchen und Interpretieren Sie die Wichtigkeit der verschiedenen Features mittels `plot(rf.OOBPermutedVarDeltaError)`.

---

<sup>2</sup>[http://www.cognotics.com/opencv/servo\\_2007\\_series/part\\_2/sidebar.html](http://www.cognotics.com/opencv/servo_2007_series/part_2/sidebar.html)

<sup>3</sup>[http://www.stat.berkeley.edu/~breiman/RandomForests/cc\\_home.htm](http://www.stat.berkeley.edu/~breiman/RandomForests/cc_home.htm)

4. **Shape Particle Filters (17 Punkte)** Wir formulieren eine Funktion, die die Kosten modelliert, ein Shape auf ein Target-Bild zu fitten. D.h. wir suchen im Parameterraum (Shape-Parameter zzgl Rotation, Skalierung, Translation) einen Punkt, der ein möglichst gut passendes Shape beschreibt, dh eine Segmentierung des Objekts.
- (a) Erstellen Sie eine Funktion `train`, die auf den Trainingsbildern den Klassifikator trainiert und das PCA Shape Modell erstellt, und eine Funktion `predict`, die auf den Testbildern mit dem Klassifikator predicted.
  - (b) Erstellen Sie eine Kostenfunktion, die zu einem Parametervektor  $p$  dem Klassifikatorergebnis auf einem Testbild einen skalaren Wert liefert, der umso kleiner wird, je besser das daraus generierte Shape auf das Klassifikatorergebnis (den Wahrscheinlichkeiten für den modellierten Knochen) passt.
  - (c) Optimieren Sie diese Funktion für jedes der Testbilder. Wir verwenden dazu eine Methode aus dem Bereich der stochastischen Optimierung, genannt Differential Evolution<sup>4</sup>, die sehr einfach ist, aber robust und schnell konvergiert. Sie wird in `optimize.m` zur Verfügung gestellt. Ein komplettes Beispiel zur Erstellung/Verwendung von Kostenfunktion/Optimierung findet sich in `optimizeDEMO.m`. Alternativ kann auch die Matlab-Funktion `ga` verwendet werden, die einen genetischen Algorithmus implementiert.
  - (d) Untersuchen Sie die Segmentiergenauigkeit Ihrer Methode (praktisch hierfür zb `boxplot`). Interpretieren Sie den Einfluss der einzelnen Schritte des Algorithmus, beschreiben Sie von Ihnen untersuchte Varianten, das Konvergenzverhalten etc.

---

<sup>4</sup>[http://en.wikipedia.org/wiki/Differential\\_evolution](http://en.wikipedia.org/wiki/Differential_evolution)