

Face Recognition mit Spark und OpenCV

Hans Dieter Pogrzeba, Jakob Kusnick, Christopher Schott

OpenCV

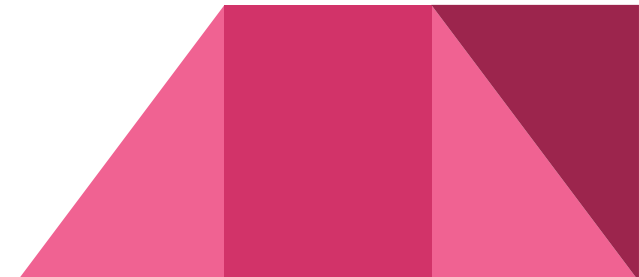


- OpenCV ist eine freie Bibliothek für die Verarbeitung von Bildern
- ursprünglich für C und C++ implementiert, auch Java-Bindings verfügbar
- ermöglicht unter anderem Face-Detection und Face-Recognition
- findet hier Anwendung im:
 - Erkennen von Augen
 - Erkennen einzelner Gesichter
 - Vorverarbeiten der Bilder
 - Erstellen von Modellen für entsprechende Gesichter

Apache Hadoop



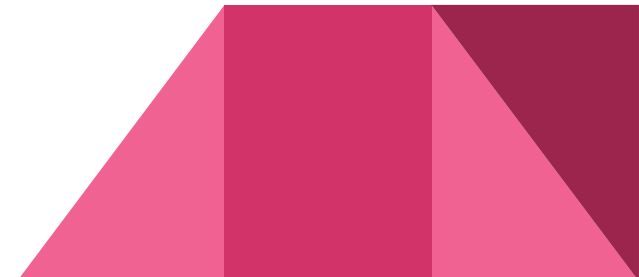
- Framework für skalierbare und verteilt arbeitende Software
- Hadoop Distributed File System
 - ermöglicht sehr große Datenmengen auf verteilten Systemen effizient zu speichern
 - ermöglicht effizienten Zugriff auf Daten
- Für uns wichtig für:
 - Speicherung der Einzelbilder von Videos
 - Speicherung der Analyseergebnisse



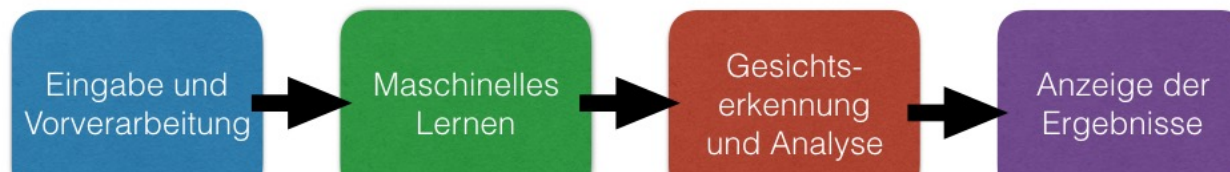
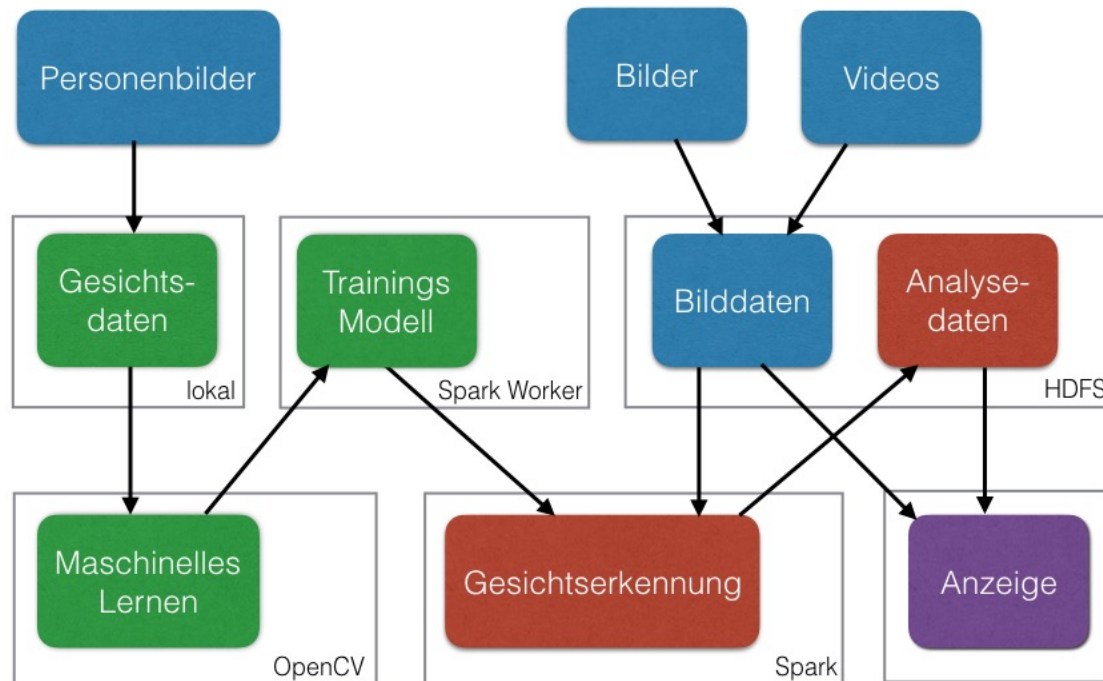
Apache Spark



- Apache Spark ist ein Framework für Cluster Computing
- Ermöglicht Arbeit mit Distributed Datasets
- Erlaubt uns:
 - parallele Verarbeitung (Erkennung) der Bildern im HDFS
 - Analyse und Aggregation der Erkennungsergebnisse als JavaRDD bzw. JavaPairRDD

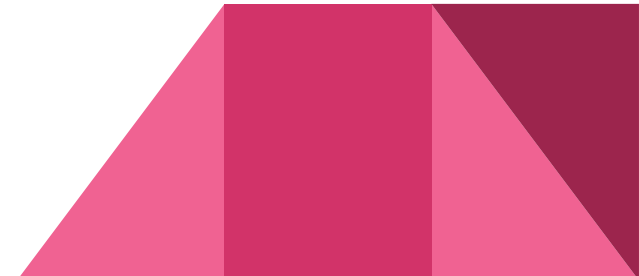


Aufbau und Ablauf



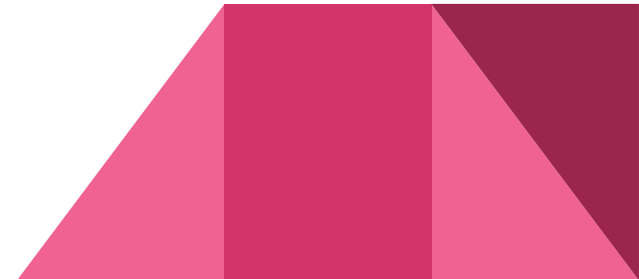
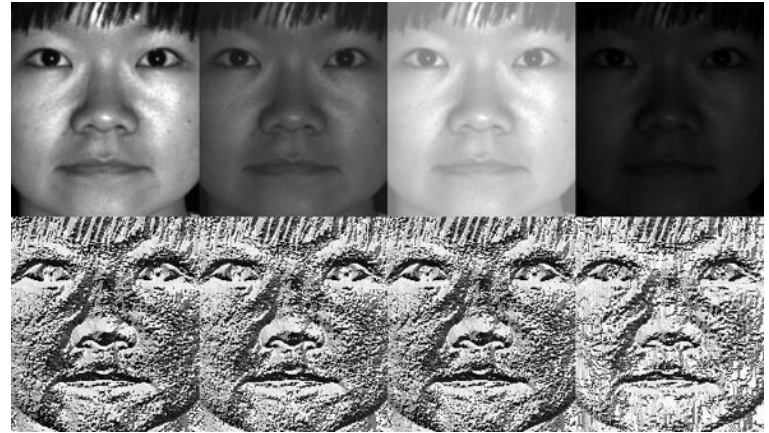
Vorverarbeitung der Bilder

- die Bilder müssen in einem OpenCV konformen Format vorliegen
- Vorverarbeitung der Bilder (Ausschneiden, Skalieren, Drehen, SW)



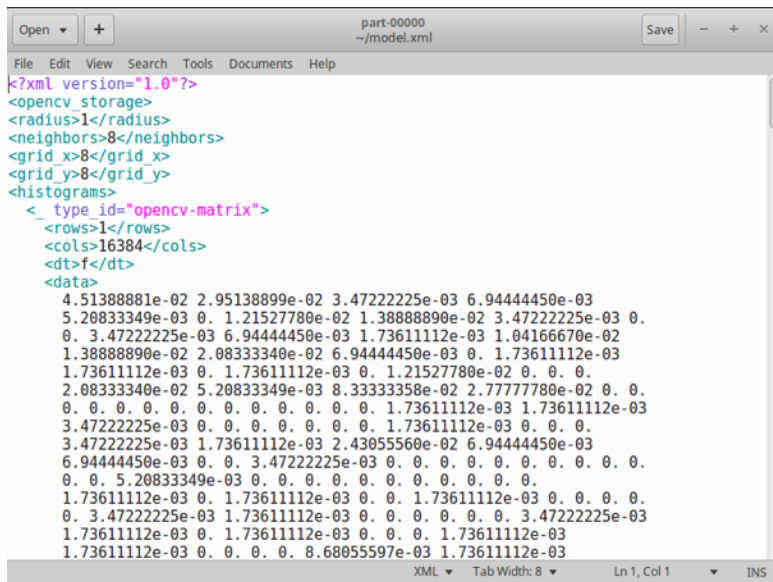
Maschinelles Lernen der Gesichter

- die vorverarbeiteten Bilder werden mittels Lernprozess von OpenCV in ein Modell umgewandelt
- Modell wird an jeden Worker verteilt
- 3 verschiedene Lern-Algorithmen:
 - [Eigenfaces](#)
 - [Fisherfaces](#)
 - [LBPH \(Local Binary Patterns Histograms\)](#)



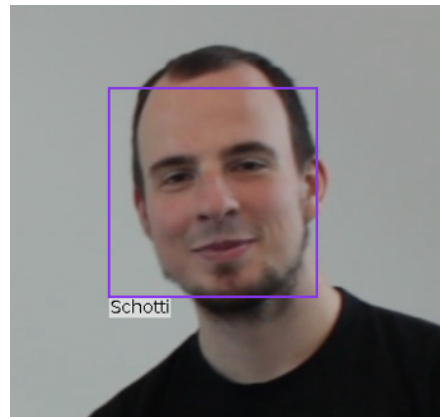
Face Recognition

- für jedes gefundene Gesicht wird mit Hilfe des Modells ermittelt, welches der erlernten Gesichter am ähnlich
- die entsprechenden Daten werden zu jedem Gesicht als JSON gespeichert und für die Nachbearbeitung mit Spark zurückgegeben



```
<?xml version="1.0"?>
<opencv_storage>
<radius>1</radius>
<neighbors>8</neighbors>
<grid_x>8</grid_x>
<grid_y>8</grid_y>
<histograms>
  <_ type id="opencv-matrix">
    <rows>1</rows>
    <cols>16384</cols>
    <dt>f</dt>
    <data>
      4.51388881e-02 2.95138899e-02 3.47222225e-03 6.94444450e-03
      5.20833349e-03 0. 1.21527780e-02 1.38888890e-02 3.47222225e-03 0.
      0. 3.47222225e-03 6.94444450e-03 1.73611112e-03 1.04166670e-02
      1.38888890e-02 2.08333340e-02 6.94444450e-03 0. 1.73611112e-03
      1.73611112e-03 0. 1.73611112e-03 0. 1.21527780e-02 0. 0. 0.
      2.08333340e-02 5.20833349e-03 8.33333358e-02 2.77777780e-02 0. 0.
      0. 0. 0. 0. 0. 0. 0. 0. 0. 1.73611112e-03 1.73611112e-03
      3.47222225e-03 0. 0. 0. 0. 0. 0. 1.73611112e-03 0. 0. 0.
      3.47222225e-03 1.73611112e-03 2.43055560e-02 6.94444450e-03
      6.94444450e-03 0. 0. 3.47222225e-03 0. 0. 0. 0. 0. 0. 0. 0.
      0. 0. 5.20833349e-03 0. 0. 0. 0. 0. 0. 0. 0. 0. 0.
      1.73611112e-03 0. 1.73611112e-03 0. 0. 1.73611112e-03 0. 0. 0. 0.
      0. 3.47222225e-03 1.73611112e-03 0. 0. 0. 0. 3.47222225e-03
      1.73611112e-03 0. 1.73611112e-03 0. 0. 1.73611112e-03
      1.73611112e-03 0. 0. 0. 8.68055597e-03 1.73611112e-03
```

```
{{"x":1219,"width":200,"y":244,"height":200,"prediction":1},
{"x":1058,"width":175,"y":321,"height":175,"prediction":2}}
```



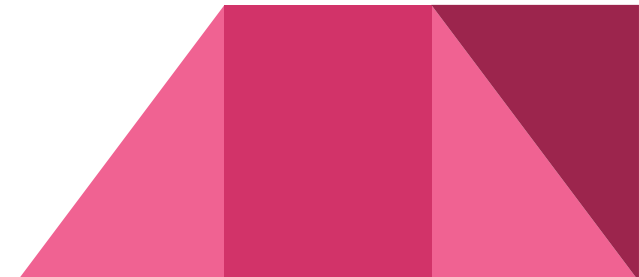
Analyseergebnisse mit Spark

- der Erkennungsprozess liefert ein JavaRDD<Tuple2<Integer, HashSet<Integer>>> von Framezahl und FaceID, das für weiterführende Analysen mit Spark benutzt werden kann
- implementierte Beispiele:
 - Zählung der Gesichter in einem (1%) Bereich des Videos:

```
2223323333333333333333232333333333333333333333333333323332233222133332333333333333333333323332222121222100000
```

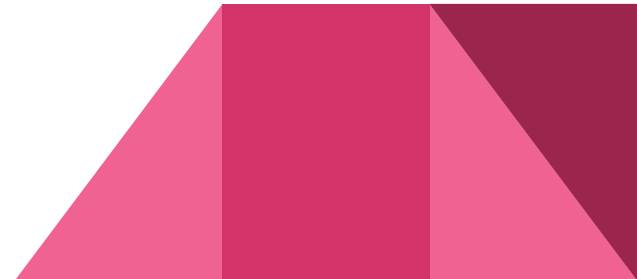
- Zählung der Frames pro Gesicht:

```
2: 499  
1: 442  
0: 360
```



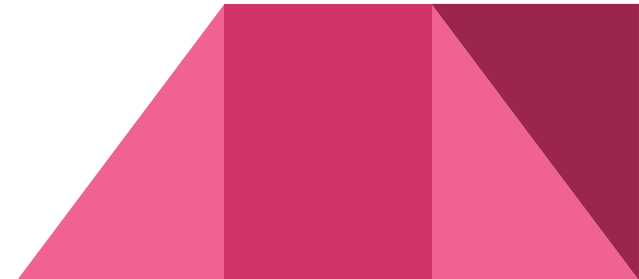
Demo

Es folgt eine kurze Demovorführung unseres Projektes



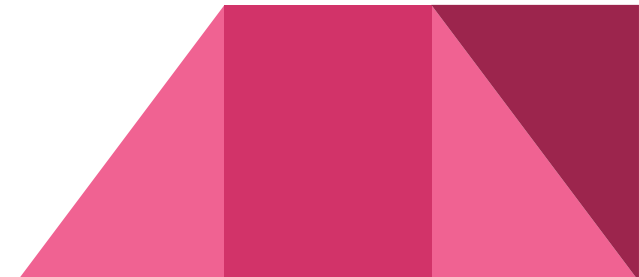
Erkenntnisse und Probleme

- Erkenntnisse:
 - Verbindung von OpenCV mit Spark ist prinzipiell möglich
- Probleme:
 - Aufgrund der fehlenden Serialisierbarkeit der Bilder, umständliche Benutzung über Dateipfad des HDFS
 - Umständliches verteilen des Modells auf die einzelnen Spark Worker
 - läuft in virtueller Maschine, jedoch ist der Betrieb auf Clustern noch fragwürdig



Weiterführende Arbeit

- Optimierung der Parameter für zuverlässigere Erkennung
- Streamverarbeitung
 - anstelle von bereits gedrehten Videos sollten auch direkt Streamdaten verwendet werden können (z.B. Bilder aus einer Webcam)
- detailliertere Statistiken mit entsprechenden Visualisierungen
 - Beispiele:
 - Welche Personen kommen wann auf Arbeit?
 - Wann machen die Personen Pause?



Vielen Dank für die Aufmerksamkeit

Bestehen noch Fragen?

