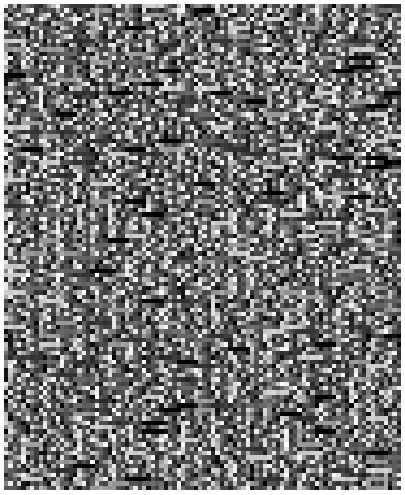
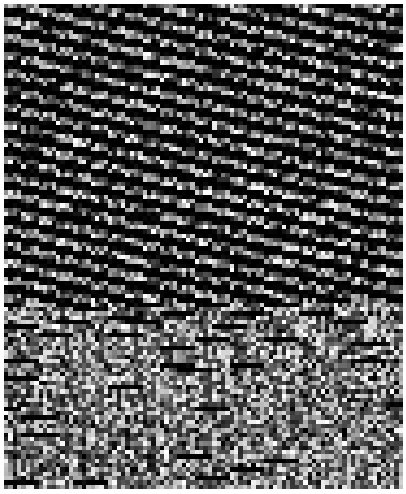
11.04.2019

- Ausarbeitung der Idee
    - Hauptkomponentenanalyse (PCA) von Porträtfotos
    - Autoencoder verwenden, um Bilder zu komprimieren
    - Durchschnittsgesicht jeder Person berechnen
    - Differenzgesichter jeder Person berechnen
    - PCA für alle Differenzgesichter aller Personen durchführen
        - ➢ Hauptkomponenten beschreiben Umgebungsfaktoren der Porträts
    - PCA für alle Durchschnittsgesichter durchführen
        - ➢ Hauptkomponenten beschreiben Eigenschaften der Gesichter
    - Eventuell mit Web-Crawler trainieren?
- Erste Versuche einer PCA in Python
  https://de.wikipedia.org/wiki/Eigengesichter#Anleitung_zur_praktischen_Implementation
    - Datensatz:
      https://www.cl.cam.ac.uk/research/dtg/attarchive/facesataglance.html
    - Laufzeit 10 Minuten
    - Ergebnisse:



Erstes berechnetes Durchschnittsgesicht

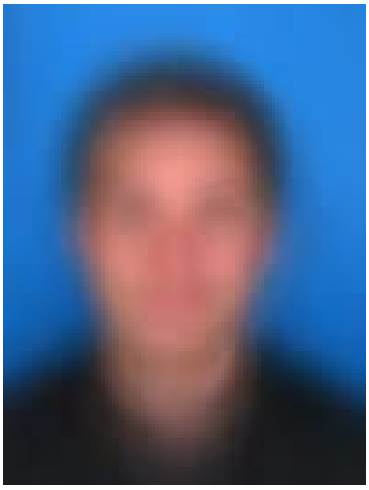


Erstes berechnetes Eigengesicht

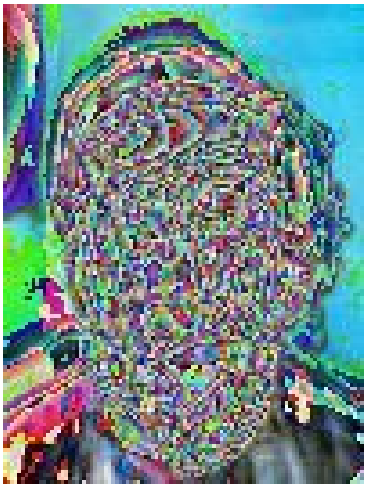
- ➢ Somethin’ aint right

12.04.2019

- Geschwindigkeitsproblem angehen
  https://de.wikipedia.org/wiki/Eigengesichter#Geschwindigkeitsproblematik
    - Dauer reduziert auf wenige Sekunden
- Fehler in der Ausgabe korrigieren (Datentyp war falsch)
- Neue Datenbank: http://pics.psych.stir.ac.uk/2D_face_sets.htm “Utrecht ECVP”
- Ergebnisse:



Durchschnittsgesicht



Ein Gesicht nach Abbildung in den niedrig dimensionalen Unterraum und Rückabbildung (Erwartet wurde, das Originalbild zu erhalten)

- Reihenfolge der Operationen vertauscht; Zuerst Eigenvektor der Hilfsmatrix L = A^T A normalisieren und dann mit Differenzmatrix A multiplizieren, um Eigenvektor der Kovarianzmatrix C = A A^T zu erhalten

- Ergebnisse: Erste fünf Eigengesichter des Utrecht-Sets in Reihenfolge

- Zweiter Versuch:

- Kodieren und Dekodieren eines Bildes erzeugt nur Rauschen:

13.04.2019

- Neue Datenbank: https://www.nist.gov/itl/iad/image-group/color-feret-database
- “Covariance-Dropoff” eingebaut (unwichtige Eigenvektoren werden verworfen)
- Berechnen der Eigenvektoren behoben
    - Berechnung war falsch, die resultierende Eigenvektor-Matrix war nicht orthogonal
    - Berechnung nach http://www.scholarpedia.org/article/Eigenfaces#Computing_the_Eigenfaces (Punkt 5)
    - Resultat
        - Eigengesichter sehen jetzt alle aus wie Durchschnitt (vermutlich weil Betrag der Eigenvektoren zu klein)

- Kodieren und Dekodieren funktioniert!

 wird zu  (Dropoff: 97%)

18.04.2019 (Labor)

- Ausgabe der PCA-Ergebnisse im Numpy Binär-Format
- Begonnen mit Entwicklung einer Website zur Darstellung
    - Javascript-Code zum Laden von Numpy-Binär-Dateien https://gist.github.com/nvictus/88b3b5bfe587d32ac1ab519fd0009607
    - Javascript-Bibliothek für Numpy-Operationen “numjs” https://github.com/numjs/numjs

25.04.2019 (Labor)

- Website weiterentwickelt
- Minimum und Maximum von Koordinaten im Gesichtsraum ermittelt und als Grenzen für Schieberegler verwendet

02.05.2019 (Labor)

- Schieberegler werden beim Laden des Durchschnittsgesichtes auf die richtigen Werte gesetzt
- Website-Layout verbessert

05.05.2019

- Website weiterentwickelt

09.05.2019 (Labor)

- Recherche CNNs
    - Downsampling?
    - Locally-Connected Layer?

23.05.2019 (Labor)

- autoencoder.py

06.06.2019 (Labor)

- autoencoder.py weiterentwickeln

12.06.2019

- Autoencoder funktioniert

-  wird zu  (50 Epochen Training)

13.06.2019 (Labor)

- Ausrichten von Bildern nach den Augen (aligner.py)
- Problem: Monochrome Bilder rausfiltern -> Singulärwertzerlegung?

17.06.2019

- aligner.py
  - Monochrome Bilder rausfiltern: https://infoscience.epfl.ch/record/33994/files/HaslerS03.pdf (7.)
  - Unscharfe Bilder herausfiltern: http://optica.csic.es/papers/icpr2k.pdf (3.3.)
  - multiprocessing library zum Generieren von mehreren Worker-Threads

20.06.2019 (Labor)

- Gruppieren von Gesichtern nach Person

23.06.2019

- Interaktive Excel-Datei zum Berechnen der Parameter und Gewichte

25.06.2019

- Trainieren mit Generator

26.06.2019

- Checkpoints beim Training

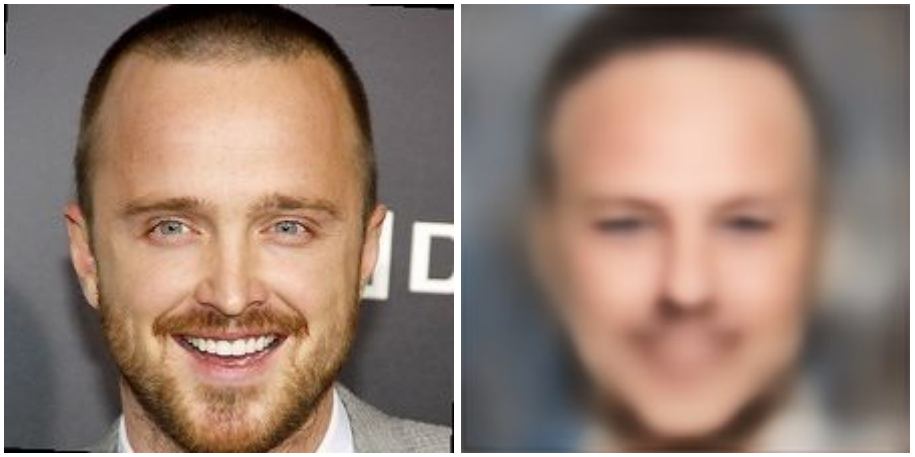
27.06.2019 (Labor)

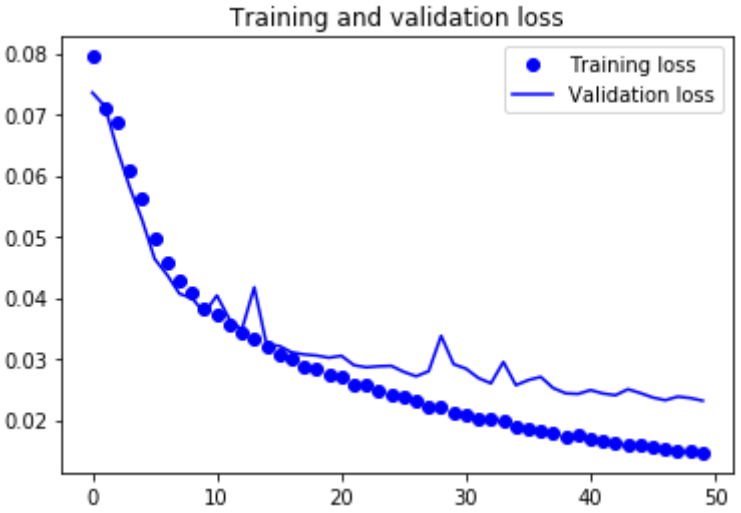
- Gruppierung von Gesichtern verbessert

04.07.2019 (Labor)

- Troubleshooting des Trainings

11.07.2019 (Labor)

- Troubleshooting des Trainings
  - Verwende SGD mit lr = 0.1
  - Verwende mean_squared_error
- Laden der VGG16 Gewichte korrigiert
  - Die Gewichte der Convolutional Layers werden übernommen
  - Alle Conv Layers bis auf die letzten drei sind trainierbar
- Training erfolgreich (~5000 Bilder, 50 Epochen):
  - 



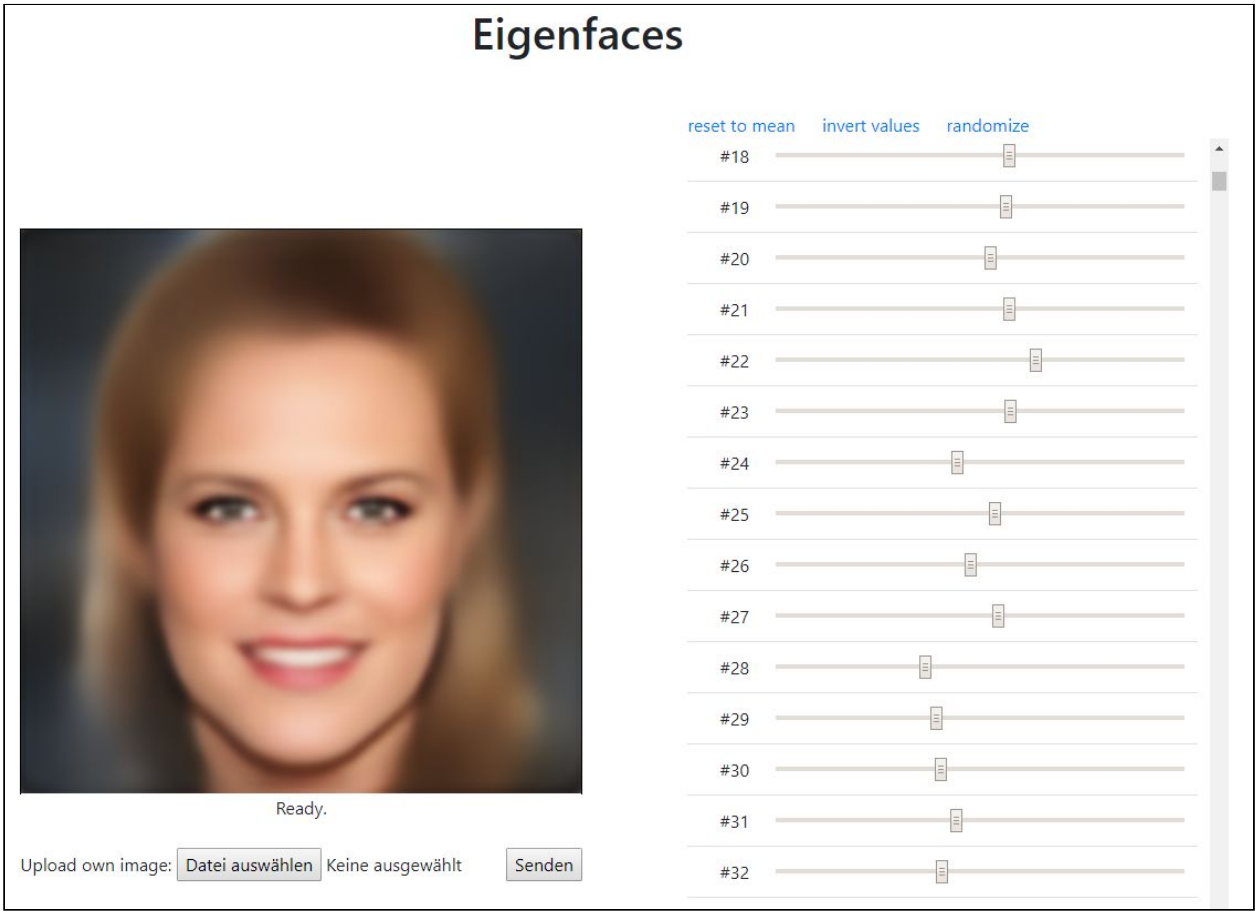
-

16.08.2019

- Verbinde autoencoder und PCA
- Flask App als Demo
- Klappt erstaunlich gut

26.08.2019

- Bisschen troubleshooting
  - Autoencoder wurde an falscher Stelle getrennt
  - PCA geändert, sodass große Datensätze verarbeitet werden können
- Hochladen von eigenen Bildern



27.08.2019

- Problem: Um echte Gesichter wiederherstellen zu können braucht man mindestens 9 Standardabweichungen nach links und rechts für jeden Eigenwert-Regler
    - Warum?
    - Zu viel overfitting?
    - Größe des bottlenecks im Autoencoder verringern?

29.08.2019

- Größe des bottlenecks von 1000 auf 200 verringern
    - Hoffentlich mehr Abstraktion, weniger overfitting
    - Netzwerk-Struktur: conv(3,3,64)>conv(3,3,64)>maxpool(2,2)>conv(3,3,128)>conv(3,3,128)>maxpool(2,2)>conv(3,3,256)>conv(3,3,256)>conv(3,3,256)>maxpool(2,2)>conv(3,3,512)>conv(3,3,512)>conv(3,3,512)>maxpool(2,2)>conv(3,3,512)>conv(3,3,512)>conv(3,3,512)>maxpool(2,2)>flatten()>dense(4096)>dense(4096)>dense(200)

03.09.2019

- Versuche bottleneck Größe 500
- Dummer Fehler: Google Colab bricht nach 12h ab, danach starte ich mit checkpoint neu. Dadurch werden die 20% der Bilder zur Validierung neu gewählt. Also sieht das Netzwerk mehr Bilder als gewünscht
    - Teile Trainingsbilder und Validierungsbilder schon vorher

04.09.2019

- Versuche bottleneck Größe 300
- Speichere Veränderung von loss und val_loss beim Training in einer .tsv Datei, um diese Daten bei Absturz nicht zu verlieren