

Camera Trap Challenge

Praktikum Computer Vision

Aufbau

1. Aufteilung auf Sequenzen
2. Segmentierung und Lokalisierung
3. Klassifizierung mit HOGs und SVMs
4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs
5. Evaluierung
6. Ausblick

1. Aufteilung auf Sequenzen

2. Segmentierung und Lokalisierung

3. Klassifizierung mit HOGs und SVMs

4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs

5. Evaluierung

6. Ausblick

Aufteilung auf Sequenzen

- ▶ sortiere Bilder nach Seriennummer der Kamera und Zeitpunkt der Aufnahme
- ▶ dafür notwendig: Zugriff auf die EXIF Maker Notes mit Hilfe des Programms „Exiftool“
[**exif**]
- ▶ unterteile die sortierte Folge von Bildern in Sequenzen, falls sich die Seriennummer ändert oder der Zeitunterschied zwischen zwei Bildern zu groß wird
- ▶ Umsetzung in Programm „Camera Trap Sequencer“:
 - ▶ setzt viele nützliche Funktionen zum Aufteilen auf Sequenzen um
 - ▶ GUI implementiert mit Qt 5

Camera Trap Sequencer

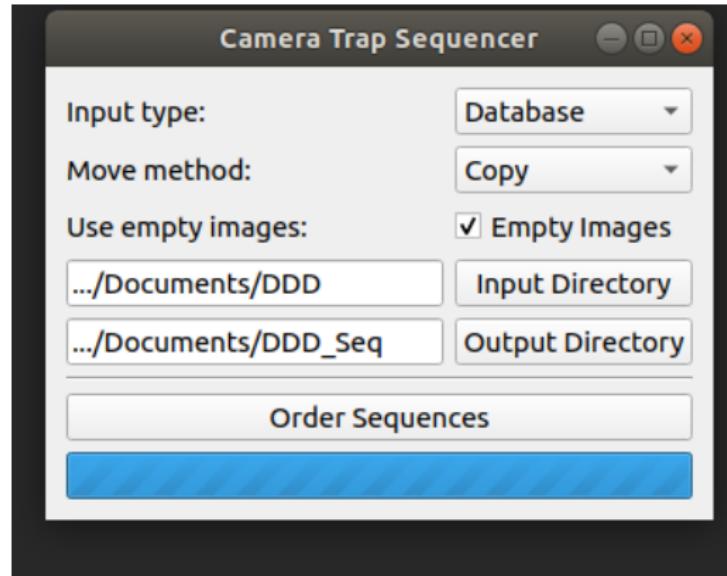


Abbildung: Grafische Benutzeroberfläche zur Aufteilung von Kamerafallensequenzen.

1. Aufteilung auf Sequenzen

2. Segmentierung und Lokalisierung

3. Klassifizierung mit HOGs und SVMs

4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs

5. Evaluierung

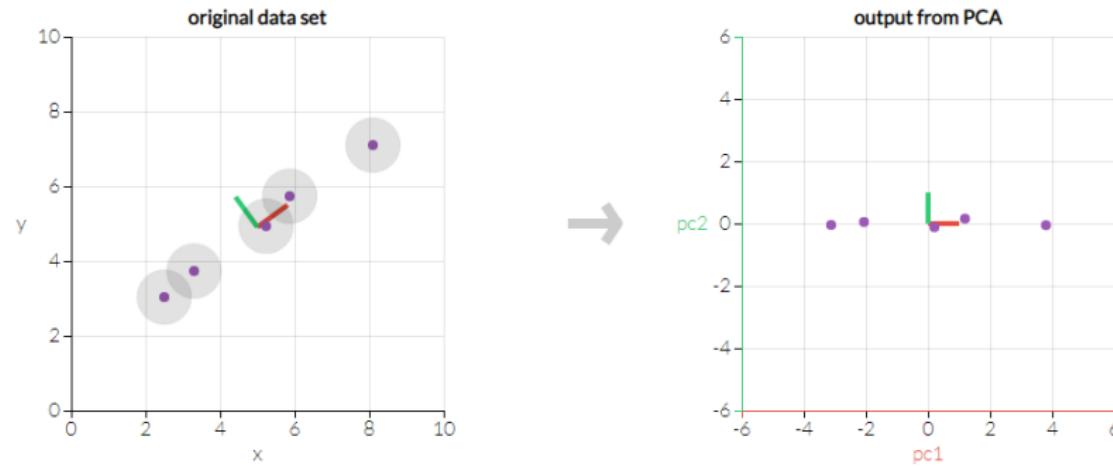
6. Ausblick

Bewegungsdetektion mit Principal Component Analysis (PCA)

- ▶ Bewegungsdetektion durch Vordergrund- und Hintergrunderkennung
- ▶ Vorgehen:
 - ▶ betrachte Datensatz als Sequenz von Bildern
 - ▶ berechne redundante Informationen (Hintergrund)
 - ▶ Den Vordergrund erhält man, indem man das Low-Rank Hintergrundbild vom Datensatz subtrahiert.

Principal Component Analysis (PCA)

- ▶ lineare Transformation der Variablen
- ▶ Projizierung in einen neuen Unterraum
- ▶ Dimensionen reduzieren



<http://setosa.io/ev/principal-component-analysis/>

Bewegungsdetektion mit PCA

- ▶ Vordergrund- und Hintergrunderkennung
 - ▶ Berechnung vom Vordergrund.
 - ▶ PCA sucht die ersten k -Hauptkomponenten, die die Daten mit einem maximalen Varianz beschreiben.
 - ▶ PCA kann mit der ersten Komponenten den Hintergrund von Sequenzen approximieren.

Bewegungsdetektion mit PCA

- ▶ Low-Rank Approximation
 - ▶ Berechne Singulärwertzerlegung aller Bilder von Sequenz X

$$\text{SVD}(X) = C = U\Sigma V^T \quad (1)$$

- ▶ Leite die Matrix Σ_k von Σ , sodass die $n - k$ Werte entlang der Diagonale durch 0 ersetzt werden.
- ▶ Dies ergibt die Low-Rank Approximation:

$$C_k = U\Sigma_k V^T \quad (2)$$

Bewegungsdetektion mit PCA

- ▶ Vordergrund- und Hintergrunderkennung
 - ▶ Berechnung vom Hintergrund:

$$L = C_1 = U\Sigma_1V^T \quad (3)$$

Originalbild



M

Hintergrund



L

Bewegungsdetektion mit PCA

- ▶ Vordergrund- und Hintergrunderkennung
 - ▶ Berechnung vom Vordergrund:



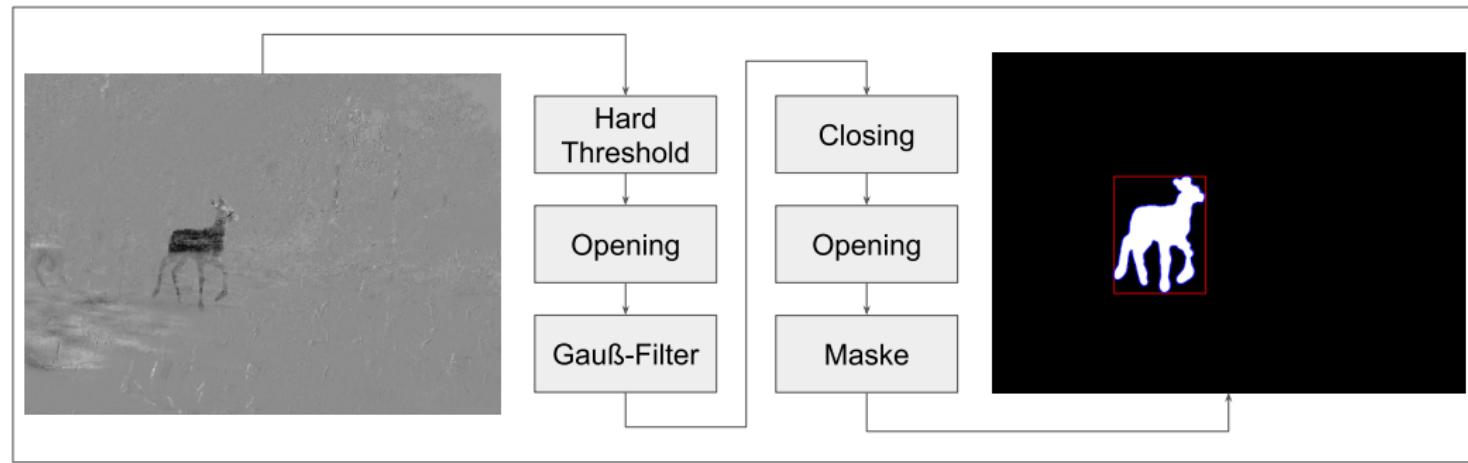
$$M - L = S$$

Bewegungsdetektion mit PCA

- ▶ Nachbearbeitung vom Vordergrund:
 - ▶ Hard- und Soft-Thresholding
 - ▶ aktive Konturen
 - ▶ Median- und Gauß-Filterung
 - ▶ Morphologische Operationen

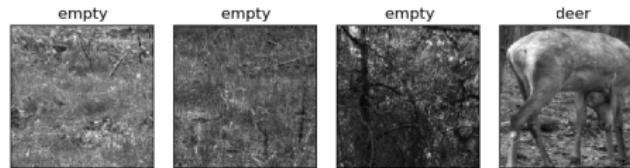
Bewegungsdetektion mit PCA

- ▶ Nachbearbeitung vom Vordergrund:



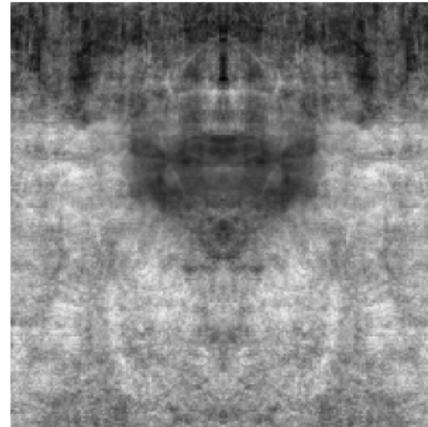
Objektdetektion mit PCA

- ▶ Vorbearbeitung der Daten:



Objektdetektion mit PCA

1. Daten zentrieren



Mittelwertbild

$$C = X - \bar{x}$$

(4)

Objektdetektion mit PCA

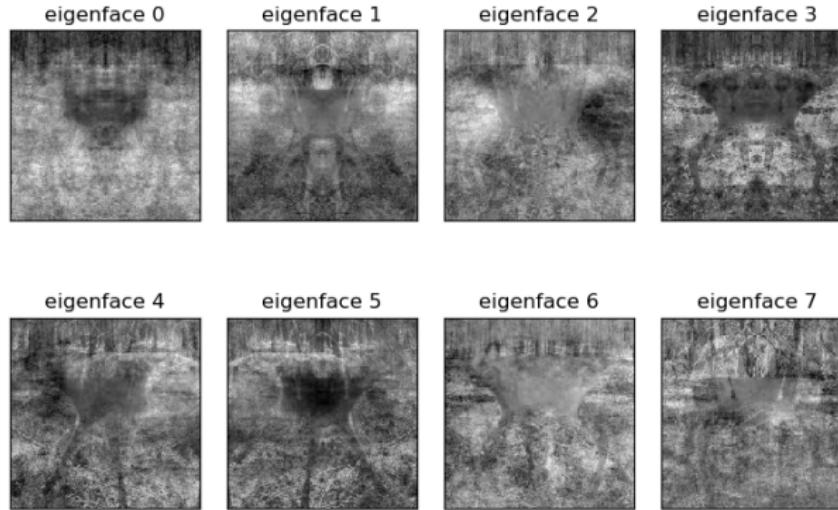
2. Berechne die Eigenwerte und Eigenvektoren für die Kovarianzmatrix CC^T :

$$\text{SVD}(C) = U\Sigma V^T \quad (5)$$

3. Projektion des Datensatzes X in den r -Unterraum:

$$Y = U_r^T(X - \bar{x}) \quad (6)$$

Objektdetektion mit PCA



Objektdetektion mit PCA

predicted: badger
true: badger



predicted: badger
true: badger



predicted: deer
true: deer



predicted: deer
true: deer



predicted: deer
true: deer



predicted: empty
true: empty



predicted: deer
true: deer



predicted: deer
true: empty



	precision	recall
--	-----------	--------

deer	0.92	0.81
badger	0.73	1.00
empty	0.67	0.60

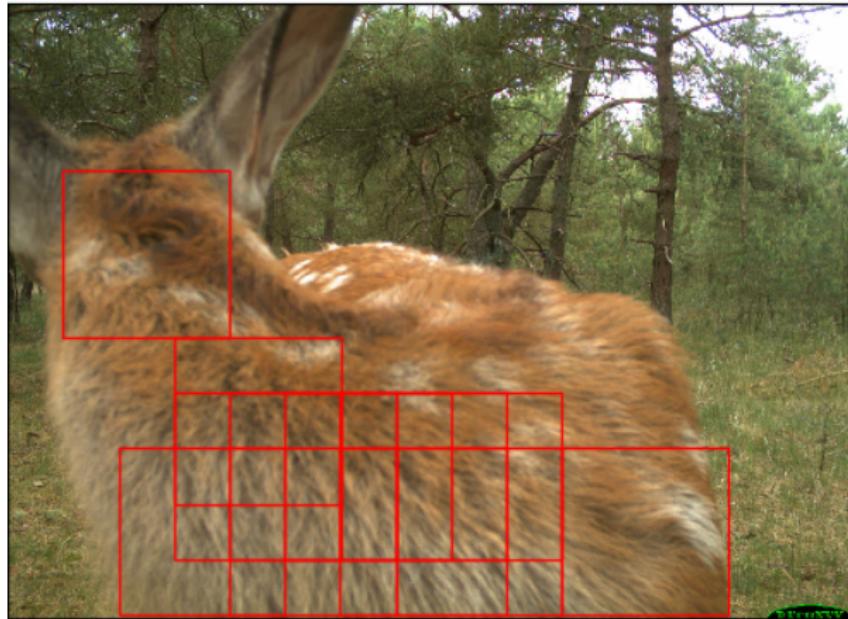
Objektdetektion mit PCA

- ▶ Sliding Window



Objektdetektion mit PCA

similar to 'deer'



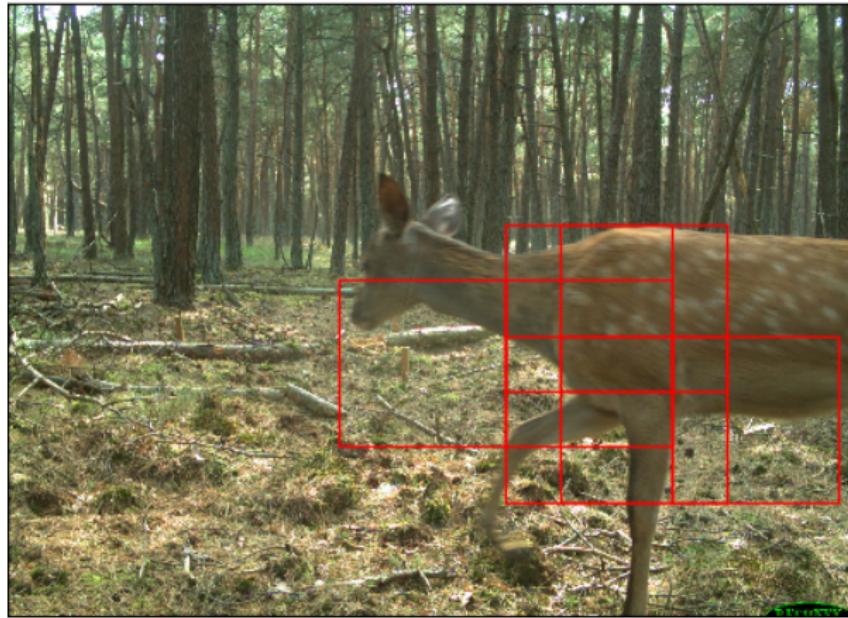
Objektdetektion mit PCA

similar to 'deer'



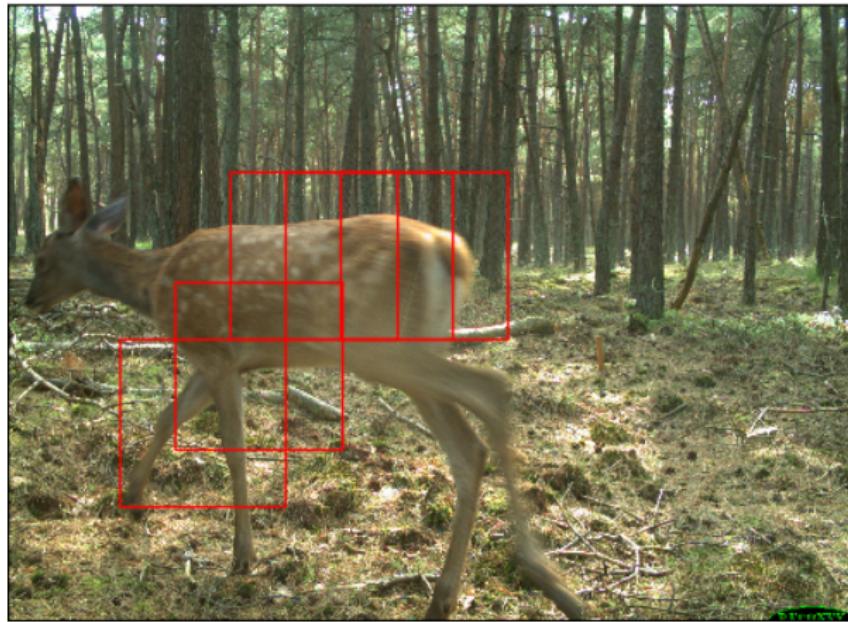
Objektdetektion mit PCA

similar to 'deer'



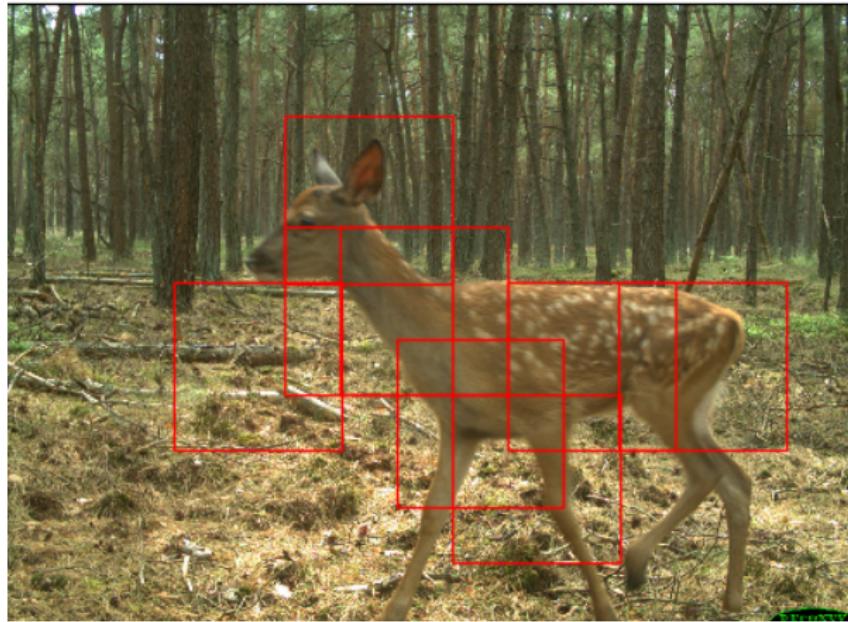
Objektdetektion mit PCA

similar to 'deer'



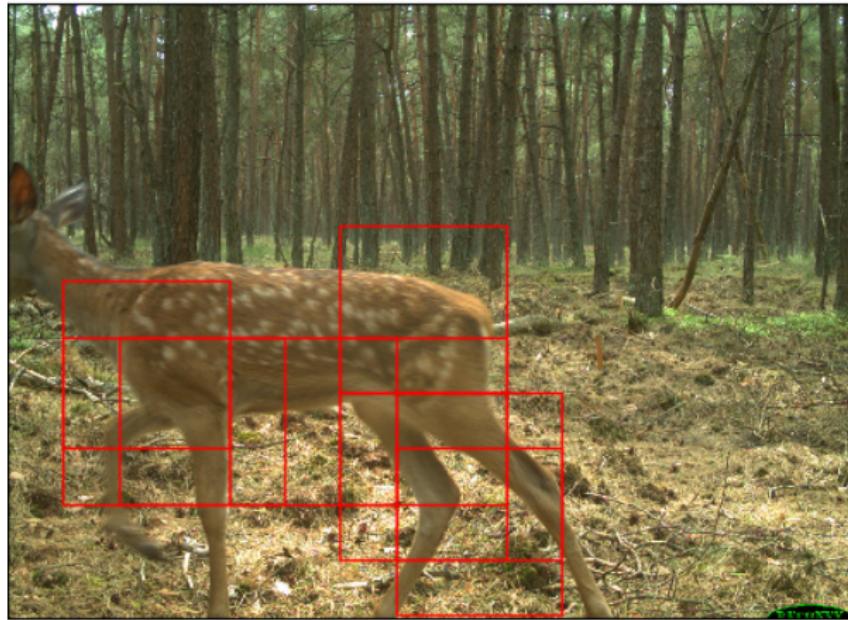
Objektdetektion mit PCA

similar to 'deer'



Objektdetektion mit PCA

similar to 'deer'



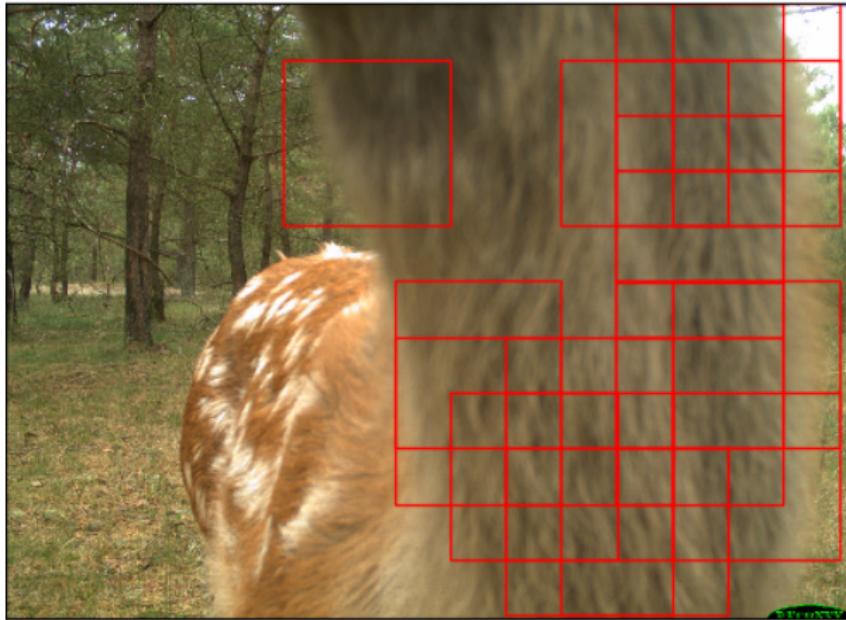
Objektdetektion mit PCA

similar to 'deer'



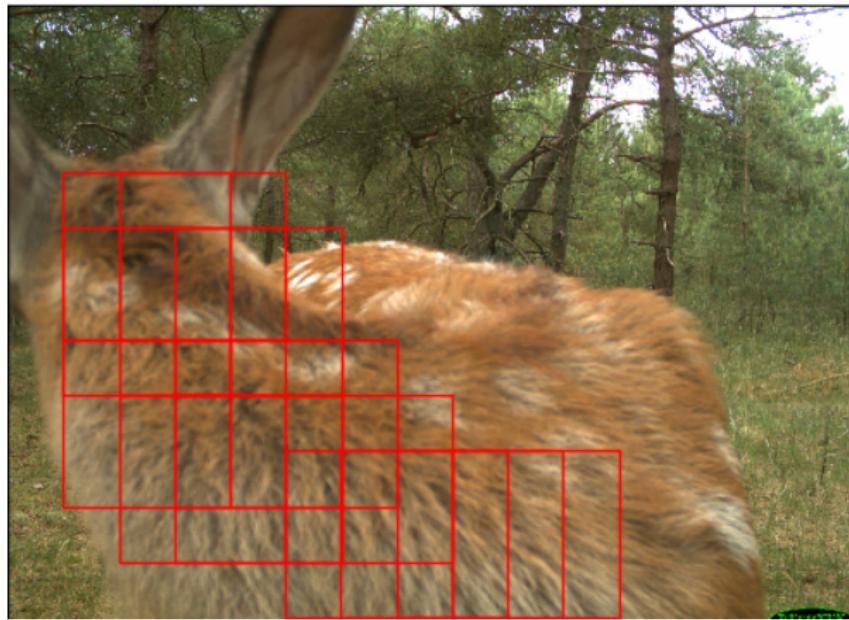
Objektdetektion mit PCA

similar to 'deer'



Objektdetektion mit PCA

similar to 'deer'



1. Aufteilung auf Sequenzen
2. Segmentierung und Lokalisierung
3. Klassifizierung mit HOGs und SVMs
4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs
5. Evaluierung
6. Ausblick

Schematischer Ablauf

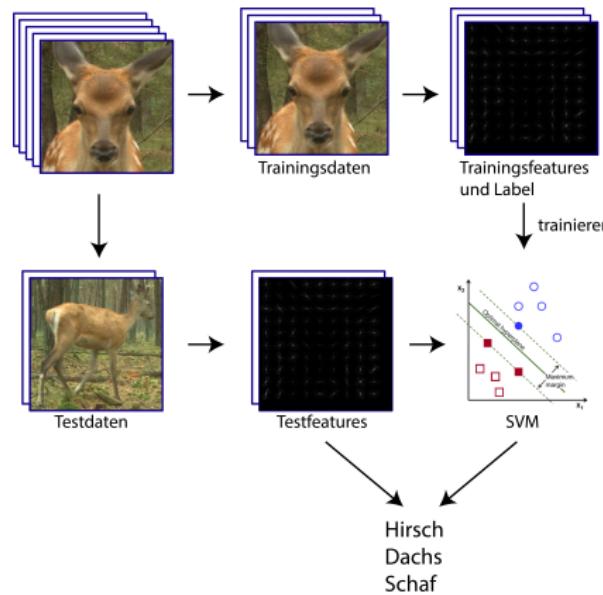


Abbildung: Übersichtsbild einer Klassifizierung

Histogram of Oriented Gradients

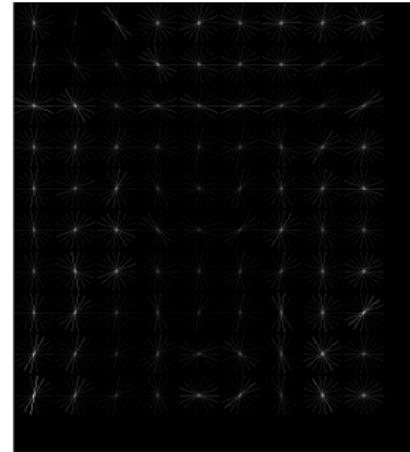


Abbildung: Visualisierung des HOG Feature Descriptors

Berechnung des HOGs

1. Vorverarbeitung (optional)
2. Gradienten Berechnung (Sobel)

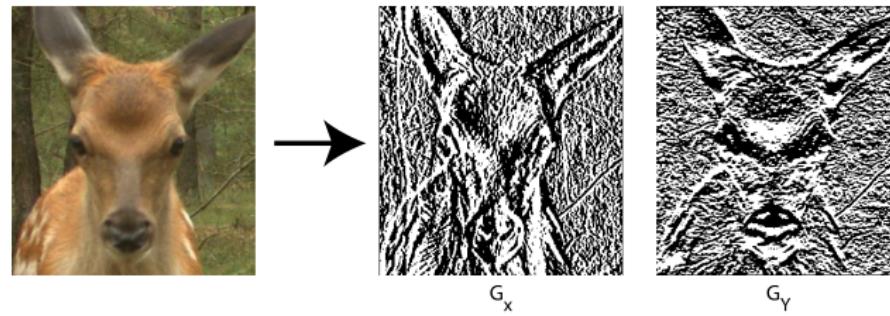


Abbildung: Visualisierung der Gradientenberechnung

$$\text{Magnitude} : |G| = \sqrt{G_x^2 + G_y^2} \quad (7)$$

$$\text{Orientierung} : \Theta = \arctan(G_x, G_y) \quad (8)$$

Berechnung des HOGs

1. Vorverarbeitung (optional)
2. Gradienten Berechnung (Sobel)
3. Unterteilung des Bildes in Rechtecke
4. Extraktion eines Histogramms für jeden Block aus 4 Rechtecken
5. Konkatenation der Histogramme zu einem gesamt Feature => HOGs

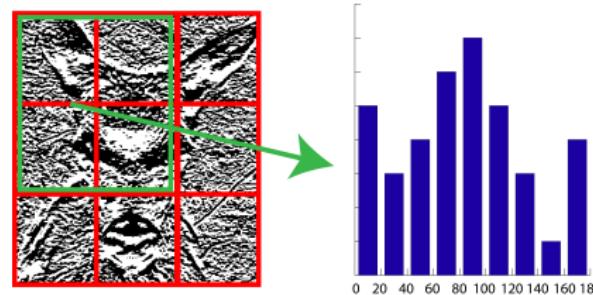


Abbildung: Berechnung eines Histogramms

Verwendete Support Vector Machine

- ▶ openCV Implementierung
- ▶ Kernel: Radiale Basisfunktion
- ▶ Bestrafung von Ausreißern
- ▶ Gewichtung eines Datenpunktes $\gamma = 0,5$
- ▶ $C = 12,5$

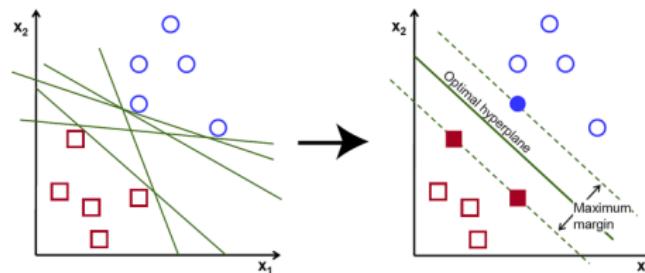


Abbildung: Beispiel für eine Klassifizierung mit einer SVM (Quelle:
https://docs.opencv.org/3.4.2/d4/db1/tutorial_py_svm_basics.html)

Testparameter und Evaluierung

- ▶ Testparameter:
 - ▶ Dachs und Hirsch Tagesbilder
 - ▶ Bis zu 66% Trainingsdaten; Rest Testdaten
 - ▶ Zusätzlich um gespiegelte Trainingsdaten erweitert

Mittlere Präzision	standard	gespiegelt
Hirsch	96,3% ($\sigma = 0.033$)	94,5% ($\sigma = 0.035$)
Dachs	81,3% ($\sigma = 0.125$)	83,7% ($\sigma = 0.109$)

Tabelle: Auswertung der durchschnittlichen Präzision des HOGs in Kombination mit einer SVM auf Testdaten mit zwei verschiedenen Labeln aus 50 Trainings- und Evaluierungs-Durchläufen. Die Parameter des HOG-Descriptors wurden zuvor optimiert.

- ▶ Beobachtung: Eine Einheitliche Trainingsdatenmenge erhöht die Präzision deutlich

1. Aufteilung auf Sequenzen
2. Segmentierung und Lokalisierung
3. Klassifizierung mit HOGs und SVMs
4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs
5. Evaluierung
6. Ausblick

Übersicht

- ▶ Ansatz zur Klassifizierung von **Isp06**
- ▶ ursprünglich zur Klassifikation von Szenen benutzt (z.B. Gebirge, Strand, Stadt)
- ▶ Erweiterung des Bag-of-Words-Ansatzes, bei dem auch die räumliche Anordnung von Features im Bild beachtet wird
- ▶ Aufbau des Algorithmus:
 1. berechne SIFT-Features über zufälligen Bildausschnitten und clustere sie in Codebook
 2. berechne für jedes Eingabebild dichte SIFT-Features in Spatial Pyramid
 3. wende Locality-constrained Linear Coding (LLC) auf jedem Feature an
 4. pool die LLC-Codes aus jedem Spatial Bin
 5. konkateniere alle gepoolten Codes der Spatial Pyramid
 6. normalisiere diese konkatenierten Codes
 7. trainiere damit eine Support Vector Machine mit linearem Kernel

Spatial Pyramid

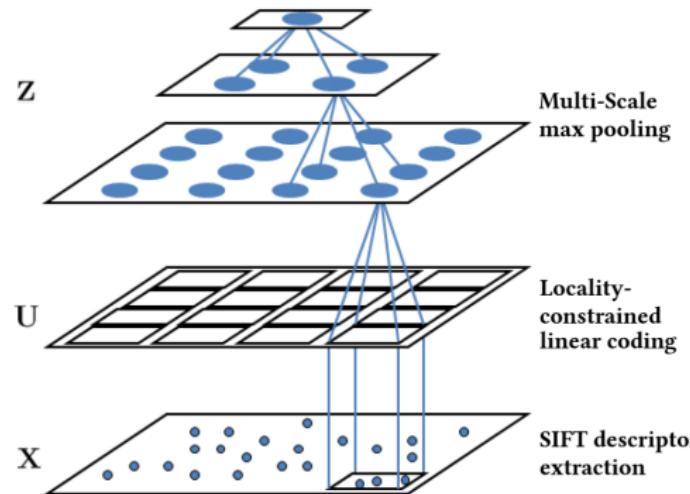


Abbildung: Architektur des Spatial-Pyramid-Matching-Ansatzes [yygh09]

Locality-constrained Linear Coding (LLC)

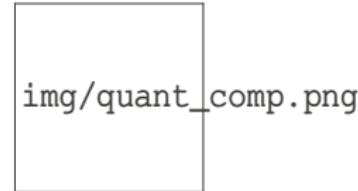


Abbildung: Vergleich verschiedener Quantisierungsstrategien [**wyylhg10**]

Locality-constrained Linear Coding (LLC) II

- ▶ Gegeben D-dimensionale Features $X = [x_1, \dots, x_N] \in \mathbb{R}^{(D \times N)}$ und Codebook mit M Einträgen $B = [b_1, \dots, b_M] \in \mathbb{R}^{(D \times M)}$.
- ▶ Gesucht: lokale Zuordnung $C = [c_1, \dots, c_N] \in \mathbb{R}^{(M \times N)}$, die Features aus X Visual Words aus B zuweist

$$\min_C \sum_{i=1}^N \|x_i - B \cdot c_i\|^2 + \lambda \cdot \|d_i \odot c_i\|^2, \text{ s.t. } 1^T c_i = 1 \forall i \quad (9)$$

- ▶ Hierbei ist \odot die elementweise Multiplikation und d_i ist ein Distanzterm, der die Ähnlichkeit von Feature x_i zu allen Visual Words des Codebooks wiederspiegelt.

$$d_i = \exp \left(\frac{\text{dist}(x_i, B)}{\lambda} \right) \quad (10)$$

Analytische Lösung von LLC

- ▶ Anders als viele Quantisierungsstrategien lässt sich LLC analytisch lösen.
 - ▶ wir können selber etwas programmieren
 - ▶ (angeblich) in der Praxis schneller

$$\text{ber. Kovarianzmatrix } C_i = (B - 1x_i^T)(B - 1x_i^T)^T$$

$$\text{Löse LGS: } (C_i + \lambda \text{diag}(d_i)) \cdot \tilde{c}_i = (1, \dots, 1)^T \quad (11)$$

$$\text{Normalisiere: } c_i = \frac{\tilde{c}_i}{1^T \tilde{c}} \quad (12)$$

Pooling und Normalisierung

- ▶ poole die LLC-Codes für jeden Spatial Bin:
 - ▶ sum pooling: $c_{out} = c_1 + \dots + c_N$
 - ▶ max pooling: $c_{out} = \max(c_1, \dots, c_N)$
- ▶ konkateniere die gepoolten LLC-Codes für jeden Spatial Bin:
 - ▶ insgesamt 21 konkatenierte LLC-Codes (Level 0: 1, Level 1: 4, Level 2: 16)
- ▶ normalisiere die konkatenierten Codes:
 - ▶ sum normalization: $c_{out} = c_{in} / \sum_j c_{in}(j)$
 - ▶ l^2 normalization: $c_{out} = c_{in} / \|c_{in}\|_2$
- ▶ in der Praxis haben sich Max-Pooling und l^2 -Normalisierung durchgesetzt [wyylhg10]

Support Vector Machine mit linearem Kernel

- ▶ Empirische Tests haben ergeben, dass Spatial-Pyramid-Matching-Codes gut linear separierbar sind und diese bessere Ergebnisse liefern als nicht-lineare Kernel [**lsp06**].
- ▶ Benutze Support Vector Machine mit linearem Kernel $k(z_i, z_j) = z_i^T z_j$ und *quadratic hinge loss*-Funktion.
 - ▶ Training in $O(N)$ und Testen in $O(1)$ statt $O(N^3)$ und $O(N)$ für nicht-lineare Mercer-Kernel.

Umsetzung

- ▶ Programmiersprache Python
- ▶ größtenteils mit *numpy* umgesetzt
- ▶ Berechnung der SIFT-Features mit OpenCV
- ▶ Clustering für Codebook mit Scikit-learns *MiniBatchKMeans* [**sklearn**]
 - ▶ deutlich schnellere Laufzeit als *KMeans*
 - ▶ identische Ergebnisse
- ▶ Scikit-Learns *LinearSVC* als Support Vector Machine [**sklearn**]
 - ▶ setzt linearen Kernel und quadratische Hinge-Loss-Funktion schon um
 - ▶ effizienter als SVC(kernel='linear')

1. Aufteilung auf Sequenzen
2. Segmentierung und Lokalisierung
3. Klassifizierung mit HOGs und SVMs
4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs
5. Evaluierung
6. Ausblick

DDD nur Tagbilder

- ▶ 70% aller Bilder zum Trainieren, 30% zum Testen

Mean Accuracy	$\alpha = 500$	$\alpha = 100$	$\alpha = 10$	$\alpha = 1$
$\sigma = 100$	94,3%	90,5%	93,3%	87,6%
$\sigma = 10$	87,6%	-	-	87,6%

Tabelle: Größe des Codebooks: 256 Features

Mean Accuracy	$\alpha = 500$	$\alpha = 100$	$\alpha = 10$	$\alpha = 1$
$\sigma = 100$	93,3%	95,7%	87,6%	87,6%
$\sigma = 10$	87,6%	-	-	87,6%

Tabelle: Größe des Codebooks: 512 Features

1. Aufteilung auf Sequenzen
2. Segmentierung und Lokalisierung
3. Klassifizierung mit HOGs und SVMs
4. Klassifizierung mit Locality-constrained Linear Coding, Spatial Pyramid Matching und linearen SVMs
5. Evaluierung
6. Ausblick

Ausblick

- ▶ PCA funktioniert nur auf Sequenzen von Bildern
 - ▶ Idee: baue während PCA Index von Bildern aller Kameras (nach Seriennummer) auf
 - ▶ benutze diesen Index zur Segmentierung von Einzelbildern, indem diese zur Sequenz vervollständigt werden
- ▶ Spatial Pyramid Matching ist momentan sehr langsam
 - ▶ Bottleneck: die Berechnung der LLC-Codes, insbesondere die Lösung des linearen Gleichungssystems
 - ▶ optimiere Code mit *numba* und parallelisiere weiter [**numba**]
- ▶ Kombiniere SPM mit LLC durch verschiedene Arten von Features, wie von [**ywkjwh13**] vorgeschlagen
 - ▶ berechne Textur- oder Farbfeatures, wie beispielsweise CLBP auf dichtem Gitter
 - ▶ Kombination mit SIFT-Features über *Boosting* oder Scikit-learns *Voting Classifier*

Quellen I