Bachelorarbeit

Klassifikation von Texturmerkmalen

Dominik Düsel

14. Oktober 2014

Prüfer: Prof. Dr. Tomas Sauer

FORWISS, Universität Passau

Inhaltsverzeichnis

Einleitung

Bildverarbeitung ist ein wichtiges Forschungsgebiet der Informatik. Ein Teilgebiet davon ist die automatische Erkennung bestimmter Objekte wie zum Beispiel eines Gesichtes oder einer Straße in einem digitalen Bild. Dafür gibt es einige verschiedene Lösungsansätze, die alle ihr Vor- und Nachteile haben.
Einer dieser Ansätze beschäftigt sich mit der Textur von Bildern und versucht diese durch verschiedene mathematische Eigenschaften zu modellieren und damit wiedererkennbar zu machen. Die Erkennung der Texturmerkmale und damit der Bildobjekte wird dabei auch automatisiert und von sogenannten Klassifikatoren durchgeführt. Dieser Ansatz soll in der vorliegenden Arbeit anhand verschiedener Methoden untersucht und für automatische Fahrbahnerkennung optimiert werden.
Dafür wird zunächst eine Einführung über das zum Testen verwendete Programm und die Bewertung der Klassifikation gegeben. Anschließend werden zwei Klassifikatoren - Support Vector Machines und Neuronale Netze - vorgestellt. Dabei wird jeweils zunächst die grundsätzliche Funktionsweise erklärt und dann verschiedene Variationsmöglichkeiten erläutert. Im nächsten Abschnitt werden dann die Berechnungen der verwendeten Texturmerkmale genauer dargestellt. Begonnen wird hier mit dem Local Binary Pattern, gefolgt von den HOG-Features und der diskreten Kosinustransformation.
Im letzten Kapitel werden schließlich die vorgestellten Methoden ausführlich getestet. Dabei werden die verschiedenen Variationsmöglichkeiten untersucht und verglichen. Anschließend werden noch ein paar weitere Experimente zur Verbesserung der Fahrbahnerekennung dargestellt.

Kapitel 1

Einführung

1.1 Schreibweisen und Bezeichnungen

1.1.1 Skalarprodukt

Das Skalarprodukt zweier n-dimensinaler Vektoren $u = (u_1, \ldots, u_n)^T$, $v = (v_1, \ldots, v_n)^T$ wird folgendermaßen dargestellt:

$$\langle u, v \rangle = u_1 v_1 + u_2 v_2 + \ldots + u_n v_n$$

1.1.2 Euklidische Norm

Die Euklidische Norm eines Vektors $v = (v_1, v_2, \ldots, v_n)^T$ wird in dieser Arbeit auf diese Weise geschrieben:

$$||v|| = \sqrt{v_1^2 + v_2^2 + \ldots + v_n^2}$$

1.1.3 Klassen

Die zwei verschiedenen Klassen des Trainings (hier „Der Punkt liegt auf der Straße“ beziehungsweise „Der Punkt liegt nicht auf der Straße“) werden oft vereinfacht mit 1 und -1 oder positiv und negativ bezeichnet.

1.1.4 LBP und DCT

Das Local Binary Pattern und die diskrete Kosinustransformation werden in dieser Arbeit mit LBP beziehungsweise DCT abgekürzt.

1.1.5 Daten

Die Wörter Datenpunkt, Datenvektor oder Punkt, im Plural auch Daten, bezeichnen ein Tupel von Zahlen, das ein Objekt (beispielsweise ein Bildauschnitt, der klassifiziert werden soll) beschreibt.

1.2 Verwendete Fehlermaße und Bewertungsalgorithmen

Da in dieser Arbeit Klassifikationen verglichen werden sollen, muss zunächst einmal festgelegt werden, wie man eine bestimmte Klassifikation bewertet. Hierfür werden im folgenden drei verbreitete Fehlermaße verwendet, nämlich die sogenannten „Precision“ und „Recall“ und das aus der Kombination der beiden vorangehenden Maße enstehende „F-measure“. Für die mathematische Definition der Fehlermaße sind folgende Begriffe nötig:

- **True positives:** Die Menge aller Punkte, die vom Klassifikator als „positiv“ eingestuft werden und dies auch wirklich sind.
- **False positives:** Die Menge aller Punkte, die vom Klassifikator als „positiv“ eingestuft werden, aber in Wirklichkeit „negativ “ sind.
- **False negatives:** Die Menge aller Punkte, die vom Klassifikator „negativ“ eingestuft werden, in Wirklichkeit aber „positiv“ sind.

$|M|$ bezeichnet im Folgenden die Kardinalität einer Menge M.

1.2.1 Precision

Precision drückt aus, ein wie großer Anteil der als „positiv“ eingestuften Daten auch wirklich „positiv“ sind.

$$\text{Precision} = \frac{|\text{True positives}|}{|\text{True positives}| + |\text{False positives}|}$$

1.2.2 Recall

Recall drückt aus, wie groß der Anteil an Daten ist, die „positiv“ sind und auch als „positiv“ eingestuft werden. Die Definition lautet wie folgt:

$$\text{Recall} = \frac{|\text{True positives}|}{|\text{True positives}| + |\text{False negatives}|}$$

1.2.3 F-measure

Das „F-measure“ kombiniert die beiden vorangegangenen Fehlermaße zu einem. Es ist folgendermaßen definiert ($\beta \geq 0$):

$$F_\beta = (1 + \beta^2) \cdot \frac{\text{Precision} \cdot \text{Recall}}{\beta^2 \cdot \text{Precision} + \text{Recall}}$$

β ist hier ein Faktor, mit dem man Precision oder Recall stärker gewichten kann. In dieser Arbeit wird ausschließlich F_1 verwendet, in dem Precision und Recall gleich stark gewichtet werden. F_1 entspricht dem harmonischen Mittel von Precision und Recall.

1.2.4 Cross-Validation

Um die Klassifikatoren nun zu bewerten wird die „k-fold Cross-Validation“ verwendet. Hierbei werden aus der Menge der vorher manuell und korrekt klassifizierten Daten k gleich große Teilmengen gebildet. Daraufhin wird der Klassifikator mit der Vereinigung von $k-1$ dieser Teilmengen trainiert, anschließend vergleicht man die Vorhersage des Klassifikators bezüglich der letzten Teilmenge mit der bekannten Klasseneinteilung und ermittelt Precision und Recall der Klassifikation.
Dieser Vorgang wird nun k mal wiederholt, so dass jede Teilmenge einmal zur Validierung verwendet wird. Nun werden die Ergebnisse gemittelt, wodurch es jeweils einen Gesamtwert für Precision und Recall gibt aus denen man dann das F_1-measure bildet. In dieser Arbeit wird durchgehend die 10-fold Cross-Validation verwendet. Der Wert 10 ist dabei ohne bestimmten Grund gewählt.

1.3 Verwendete Testumgebung

Um die Klassifikatoren zu trainieren und die Texturmerkmale zu berechnen wurde ein C++-Programm verwendet. Das Programm liest Videos und Informationen, wo sich auf diesen Videos die Fahrbahn befindet, ein. Daraus können dann die (positiven und negativen) Testdaten berechnet und damit der ausgewählte Klassifikator trainiert werden. Des weiteren gibt es Funktionen zur Berechnung von Precision, Recall und des F_1-measure sowie zur Durchführung einer Cross-Validation mit den berechneten Daten.
Die beiden Klassifikatoren stammen hierbei aus der OpenCV-Bibliothek (siehe [11], [12]). Die Berechnungen der Texturmerkmale LBP und DCT wurden manuell implementiert, für die Berechnung der HOG-Features wird auch die OpenCV-Bibliothek verwendet.

Kapitel 2

Klassifikatoren

In diesem Kapitel werden zwei verbreitete Methoden zum automatischen Klassifizieren von Daten vorgestellt und untersucht. Ziel der Untersuchung ist es, die Parameter der Klassifikatoren jeweils im Hinblick auf die Fahrbahnerkennung zu optimieren. Begonnen wird mit der Erläuterung von Support Vector Machines, gefolgt von den neuronalen Netzen.

2.1 Support Vector Machines

Das Ziel von Support Vector Machines ist es, die Hyperebene zu finden, die den minimalen Abstand zu den verschiedenen Klassen maximiert. Der Abstand des nächsten Punktes der einen Klasse zur Ebene soll also genauso groß sein, wie der Abstand des zur Ebene nächsten Punktes der anderen Klasse. Allerdings kann so eine lineare Trennebene oft nicht gefunden werden, da sich die Datenpunkte der Klassen teilweise vermischen. Deswegen werden die Punkte anhand von Kernelfunktionen in einen höherdimensionalen Raum projiziert um dort nach einer entsprechenden Trennung zu suchen.

2.1.1 Mathematische Funktionsweise

Nun wird zunächst davon ausgegangen, dass eine exakte Trennung der Datenpunkte existiert. Dies ist in der Realität selten der Fall, daher wird das ganze anschließend auf eine allgemeine Version erweitert.
Die Hyperebene H wird durch ihren Normalenvektor w und ihrer Verschiebung b definiert (nach [8]):

$$H = \{x | \langle x, w \rangle + b = 0\}$$

Zu welcher Klasse ein Datenpunkt p gehört, wird durch die Auswertung von $\langle p, w \rangle + b$ bestimmt. Ist dieser Wert größer als 0, wird der Datenpunkt der einen Klasse zugeordnet, andernfalls der anderen. Abbildung 2.1 zeigt ein Beispiel für die Trennung von Datenpunkten zweier Klassen durch eine Gerade im Zweidimensionalen. Im Folgenden sei die Klasse, in der sich ein Punkt p befindet, mit $c(p)$ bezeichnet und es sei P die Menge aller Trainingspunkte.

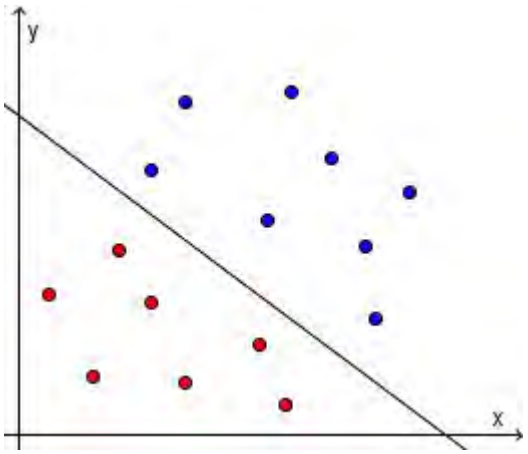

Abbildung 2.1: Trennung der Datenpunkte durch eine Hyperebene im Zweidimensionalem

Die Parameter w und b der Ebene H sind nicht eindeutig bestimmt sind, da (für beliebiges s) gilt:

$$\langle x, w\rangle + b = 0 = \langle x, s \cdot w\rangle + s \cdot b$$

Um eindeutige w, b zu erhalten, wird die Hyperebene so skaliert, dass gilt:

$$\min_{p \in P, c(p)=1} (\langle p, w\rangle + b) = 1 \tag{2.1}$$

$$\min_{p \in P, c(p)=-1} (\langle p, w\rangle + b) = -1 \tag{2.2}$$

2.1.2 Training

Ziel des Trainings ist es, die optimale Hyperebene zu bestimmen, also die Hyperebene, die zu den nächsten Punkten der beiden Klassen den maximalen Abstand besitzt. Der Abstand eines Punktes p zur Ebene H beträgt (nach [8]):

$$d(p) = \frac{1}{||w||} |\langle p, w\rangle + b|$$

Nun gilt mit Gleichung 2.1 und 2.2 für den Abstand der Punkte q_1, q_{-1} die jeweils aus ihren Klassen (1 bzw. -1) am nächsten an der Hyperebene liegen (diese beiden Punkte bezeichnet man als Support Vektoren):

$$d(q_1) = d(q_{-1}) = \frac{1}{||w||}$$

Dieser Abstand soll maximiert, beziehungsweise $||w||$ minimiert werden (mit der Nebenbedingung $c(p)(\langle p, w\rangle + b) = 1$ für $p = q_1$ und $p = q_{-1}$). Das wird mit dem Lagrange-Ansatz gemacht (mit $n = |P|$, $p_i \in P$):

$$L(w, b, \alpha) = \frac{||w||^2}{2} - \sum_{i=1}^{n} \alpha_i (c(p_i)(\langle w, p_i\rangle + b) - 1)$$

Die $\alpha_i \geq 0$ sind die Lagrange-Multiplikatoren.
Das Ziel ist nun $L(w, b, \alpha)$ zu minimieren. Nach [5, S. 263] kann dieses Minimierungsproblem auch als Minimierung folgender Funktion dargestellt werden:

$$L(\alpha) = \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j c(p_i) c(p_j) \langle p_j, p_i \rangle - \sum_{i=1}^{n} \alpha_i$$

mit den Nebenbedingungen:

$$\sum_{i=1}^{n} c(p_i)\alpha_i = 0 \qquad \forall i \in \{1, 2, \ldots, n\} : 0 \leq \alpha_i$$

Dieses Optimierungsproblem kann man z. B. mit Quadratischer Programmierung lösen. Das Verfahren dazu wird in Kapitel 2.1.4 beschrieben.

2.1.3 Erweiterung auf nicht exakt trennbaren Datenmengen

Das soll jetzt auf einen allgemeineren Fall erweitert werden. Da auch für nicht exakt trennbare Datenmengen ein Klassifikator gefunden werden soll, muss es also möglich sein, eine teilweise falsche Klassifikation von Datenpunkten zuzulassen. Dafür werden zusätzliche Variablen ξ_i eingeführt, so dass:

$$\forall i \in \{1, \ldots, n\} : c(p_i)(\langle w, p_i \rangle + b) \geq 1 - \xi_i$$

Falls nun ein Datenpunkt falsch klassifiziert wird, haben $c(p_i)$ und $\langle w, p_i \rangle + b$ unterschiedliche Vorzeichen, daraus folgt $\xi_i > 1$. Also ist $\sum_{i=1}^{n} \xi_i$ eine Art Maß für die Anzahl und Stärke der Einordnungsfehler. Dies wird nun, mit einem positiven Faktor C, zur zu minimierenden Zielfunktion addiert, um falsche Klassifikation zu bestrafen. Es soll folgende Funktion minimiert werden:

$$\frac{||w||^2}{2} + C \sum_{i=1}^{n} \xi_i$$

mit den Nebenbedingungen ($\forall i \in \{1, \ldots, n\}$):

$$c(p_i)(\langle w, p_i \rangle + b) \geq 1 - \xi_i$$
$$\xi_i \geq 0$$

Mithilfe der Lagrange-Funktion ergibt sich nach [8] ein sehr ähnliches Optimierungsproblem zu vorher:

$$\min L(\alpha) = \frac{1}{2} \sum_{i,j=1}^{n} \alpha_i \alpha_j c(p_i) c(p_j) \langle p_j, p_i \rangle - \sum_{i=1}^{n} \alpha_i$$

mit den Nebenbedigungen:

$$\sum_{i=1}^{n} c(p_i)\alpha_i = 0 \qquad \forall i \in \{1, 2, \ldots, n\} : 0 \leq \alpha_i \leq C$$

Das wird in OpenCV laut Dokumentation mit dem in [3] beschriebenen Algorithmus, nämlich mit dem „Sequential Minimal Optimization“ Verfahren, gelöst.

2.1.4 Lösung des Optimierungsproblems

Die Idee dieses Optimierungsalgorithmus ist es, das Optimierungsproblem in viele, kleinere Probleme zu teilen und dann diese zu lösen. Basierend auf einer Iteration, modifiziert man in jedem Schritt nur zwei Variablen, wodurch man in jedem Schritt auch nur nach zwei Variablen optimieren muss.

Iterationsschritte

Man braucht zunächst einen Anfangszustand für $\alpha = (\alpha_1, \alpha_2, \ldots, \alpha_n)$. Dafür wählt man den Nullvektor. Dieser Anfangszustand sei mit α^1 bezeichnet und sei $k = 1$. Nun wiederholt man folgende Schritte:

- Wenn α^k eine Extremstelle ist, wird abgebrochen (Darauf, wie man das überprüft, wird später noch eingegangen). Ansonsten wählt man zwei Punkte i, j (genaues Auswahlverfahren in [3, S. 13]) und bildet $B = \{i, j\}$ und $N = \{1, \ldots, n\} \setminus B$. Seien $\alpha_B^k = (\alpha_i, \alpha_j)$ und $\alpha_N^k = (\alpha_{n_1}, \alpha_{n_2}, \ldots, \alpha_{n_x})$ mit $\{n_1, \ldots n_x\} = N$.

- Sei $a_{ij} = \langle p_i, p_i \rangle + \langle p_j, p_j \rangle - 2\langle p_i, p_j \rangle$.
  Wenn $a_{ij} > 0$ ist, dann wird folgendes Teilproblem gelöst:

$$\begin{aligned}\min_{\alpha_i, \alpha_j} \quad & \frac{1}{2} \sum_{r,s \in B} [\alpha_r \alpha_s c(p_r) c(p_s) \langle p_r, p_s \rangle] + \sum_{s \in N} [\alpha_i \alpha_s^k c(p_i) c(p_s) \langle p_i, p_s \rangle] \\ & + \sum_{s \in N} [\alpha_j \alpha_s^k c(p_j) c(p_s) \langle p_j, p_s \rangle] - \alpha_i - \alpha_j\end{aligned}$$

  Andernfalls wird dieses Teilproblem gelöst (mit $\tau > 0$ als Konstante):

$$\begin{aligned}\min_{\alpha_i, \alpha_j} \quad & \frac{1}{2} \sum_{r,s \in B} [\alpha_r \alpha_s c(p_r) c(p_s) \langle p_r, p_s \rangle] + \sum_{s \in N} [\alpha_i \alpha_s^k c(p_i) c(p_s) \langle p_i, p_s \rangle] \\ & + \sum_{s \in N} [\alpha_j \alpha_s^k c(p_j) c(p_s) \langle p_j, p_s \rangle] - \alpha_i - \alpha_j + \frac{\tau - a_{ij}}{4} ((\alpha_i - \alpha_i^k)^2 + (\alpha_j - \alpha_j^k)^2)\end{aligned}$$

  Jeweils mit folgenden Nebenbedingungen:

$$\begin{aligned}& 0 \leq \alpha_i, \alpha_j \leq C \\ & c(p_i)\alpha_i + c(p_j)\alpha_j = -\sum_{s \in N} c(p_s)\alpha_s\end{aligned}$$

- k wird um 1 erhöht und es wird α_i^{k+1} und α_j^{k+1} auf die optimale Lösung des ausgewählten Teilproblems gesetzt, der Rest von α^{k+1} bleibt gleich α^k

Für das Verfahren zur Optimierung der Teilprobleme sei auf [3, S. 26] verwiesen.

Abbruchbedingung

Die Iteration bricht ab, sobald eine Extremstelle gefunden wurde. Nach [3, S.12], liegt eine Extremstelle genau dann vor, wenn ein b existiert, so dass:

$$m(\alpha) \leq b \leq M(\alpha) \tag{2.3}$$

mit $\nabla L(\alpha)$ als Gradient von $L(\alpha)$ und:

$$\begin{aligned}
m(\alpha) &= \max_{i \in I_{up}(\alpha)} -c(p_i)\nabla_i L(\alpha) \\
M(\alpha) &= \min_{i \in I_{low}(\alpha)} -c(p_i)\nabla_i L(\alpha) \\
I_{up}(\alpha) &= \{t \mid (\alpha_t < C \wedge c(p_t) = 1) \vee (\alpha_t > 0 \wedge c(p_t) = -1)\} \\
I_{low}(\alpha) &= \{t \mid (\alpha_t < C \wedge c(p_t) = -1) \vee (\alpha_t > 0 \wedge c(p_t) = 1)\}
\end{aligned}$$

Eine Extremstelle existiert also genau dann, wenn $m(\alpha) \leq M(\alpha)$. Da die exakte Extremstelle im Normalfall nicht erreicht wird, stoppt man, sobald

$$m(\alpha^k) - M(\alpha^k) \leq \varepsilon$$

mit einem Abbruchschwellwert $\varepsilon > 0$.

Berechnung der optimalen Hyperebene

Nachdem das vorangegangene Optimierungsproblem gelöst wurde, bekommt man den für die Hyperebene notwendigen Parameter w auf folgende Weise:

$$w = \sum_{i=1}^{n} c(p_i)\alpha_i p_i$$

Die Berechnung von b hängt von α ab. Wenn ein α_i mit $0 < \alpha_i < C$ existiert, dann folgt aus der Ungleichung (2.3):

$$b = -c(p_i)\nabla_i L(\alpha)$$

Falls kein entsprechendes α_i existiert, wird der Mittelpunkt des aus (2.3) folgenden Intervals für b verwendet:

$$b = \frac{m(\alpha) + M(\alpha)}{2}$$

2.1.5 Kernelfunktionen

Es ist oft einfacher, die Punkte in einem Raum zu trennen, der mehr Dimensionen besitzt als der Datenraum D. Sei ϕ eine Funktion, die die Datenpunkte in diesen höherdimensionalen Raum V abbildet. Nun sollen, statt der Datenpunkte x, die Punkte $\phi(x)$ getrennt werden. Um die Berechnung der Skalarprodukte in V zu vereinfachen, verwendet man

sogenannte Kernel Funktionen $\kappa : D \times D \to \mathbb{R}$, die sich wie ein Skalarprodukt in V verhalten:

$$\forall p, q \in D : \kappa(p, q) = \langle \phi(p), \phi(q) \rangle$$

Diese Funktion wird nun bei allen Berechnungen verwendet, bei denen bisher ein Skalarprodukt verwendet wurde. Wenn man dies mit der ϕ-Funktion machen würde, müsste man die Punkte in den höherdimensionalen Raum projizieren, dort das Skalarprodukt berechnen, und die Punkte wieder in D projizieren. Die Verwendung von Kernelfunktionen ist hier deutlich effizienter, da man dadurch die Berechnungen alle im niedrigerdimensionalen Raum ausführen kann und sich die Projektionen spart.
Nun eine Übersicht über verbreitete Kernelfunktionen (nach [11]):

- Lineare Kernelfunktion: $\forall p, q \in D : \kappa(p, q) = \langle p, q \rangle$
- Polynomiale Kernelfunktion: $\forall p, q \in D : \kappa(p, q) = (\gamma \langle p, q \rangle + c_0)^d$ mit $c_0 \in \mathbb{R}_0^+$, $\gamma \in \mathbb{R}^+$
- Radial Basis Kernelfunktion: $\forall p, q \in D : \kappa(p, q) = \exp(-\gamma \|p - q\|^2)$ mit $\gamma \in \mathbb{R}^+$
- Sigmoid Kernelfunktion: $\forall p, q \in D : \kappa(p, q) = \tanh(-\gamma \langle p, q \rangle + c_0)$ mit $c_0 \in \mathbb{R}_0^+$, $\gamma \in \mathbb{R}^+$

2.2 Neuronale Netze

Ein anderer Weg um Daten automatisch zu klassifizieren, sind (künstliche) neuronale Netze. Die Idee besteht darin, ein Neuronennetz, wie es z. B. in einem Gehirn vorkommt, zu simulieren. Alle neuronalen Netze sind in mehrere Schichten von Neuronen aufgeteilt, die teilweise miteinander verbunden sind. Im Folgenden soll nur eine Untergruppe der neuronalen Netze betrachtet werden, nämlich sogenannte Feedforward-Netze. Bei diesen Netzen hängt der Wert eines Neurons immer nur von den Werten der Neuronen der vorhergehenden Schicht ab. Abbildung 2.2 zeigt ein solches, dreischichtiges neuronales Netz. Es besteht aus einem „Input Layer“ mit zwei Neuronen, einem „Hidden Layer“ mit drei Neuronen und einem „Output Layer“ mit einem Neuron.
Allgemein bestehen neuronale Netze immer aus einem Input und einem Output Layer, die Anzahl der Hidden Layer ist beliebig und kann auch Null sein. Die Anzahl der Neuronen des Input Layers entspricht der Dimension der Daten, die klassifiziert werden sollen, die Anzahl der Neuronen in den Hidden Layers ist beliebig, aber offensichtlich größer als Null. Das Output Layer kann aus mehrere Neuronen bestehen, solche werden aber in dieser Arbeit nicht verwendet, da die hier gesuchten Klassifizierungen nur einen Ausgabewert benötigen, nämlich die Wahrscheinlichkeit, dass es sich bei dem Punkt um einen Teil der Straße handelt.

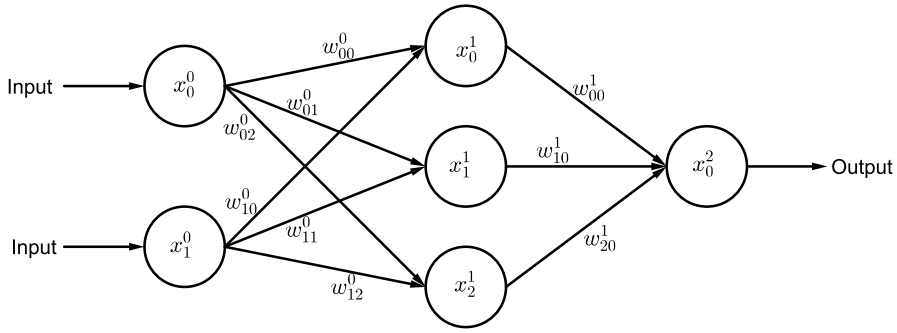

Abbildung 2.2: dreischichtiges neuronales Netz

2.2.1 Funktionsweise

Sei x_i^k das i-te Neuron in der k-ten Schicht. Der Wert der Neuronen x_i^{k+1} für $k \geq 0$ (die Schichten werden, wie in der Informatik üblich, von Null an aufwärts nummeriert) hängt von den Werten der Neuronen x_j^k in der vorherigen Schicht, deren Gewichtungen w_{ji}^k und der Aktivierungsfunktion f ab (mit d_k als Anzahl der Neuronen in der k-ten Schicht):

$$x_i^{k+1} = f(\sum_{j=0}^{d_k} x_j^k w_{ji}^k)$$

Als Aktivierungsfunktion können beispielsweise die Identität oder die Signumfunktion verwendet werden, auf eine geeignete Wahl wird nachher noch genauer eingegangen.
Nun kann man schrittweise die Werte aller Neuronen aller Schichten für einen gegeben Inputvektor berechnen. Der Wert des Outputneurons und die Kenntnis der Aktivierungsfunktion geben nun Aufschluss darüber, in welcher Klasse sich der Inputvektor befindet. Bei Verwendung der Signumfunktion z.B. entspricht der Wert des Outputneurons direkt der Klasse (-1 oder 1).

2.2.2 Aktivierungsfunktion

Als Aktivierungsfunktion f können grundsätzlich sehr viele verschiedene Funktionen verwendet werden, eine geeignete Wahl sollte allerdings einige Eigenschaften erfüllen (nach [5, S. 307]). An die Aktivierungsfunktion werden folgende Anforderungen gestellt:

- Nichtlinearität: Wäre f linear, könnten viele Sachverhalte (wie z.B. das XOR-Problem, siehe [5, S. 285]) nicht dargestellt werden.
- Beschränktheit: Dadurch bleiben die Ausgabewerte in einem abschätzbaren Bereich und können leichter in eine Klassifikation überführt werden.

- Differenzierbarkeit: Bei vielen Trainingsalgorithmen wird die Ableitung von f benötigt, um diese Algorithmen anwenden zu können muss f also differenzierbar sein.

Eine weit verbreitete Aktivierungsfunktion, die alle diese Eigenschaften erfüllt, ist die Sigmoid Funktion:

$$f(x) = a \cdot \tanh(b \cdot x)$$

Da OpenCV außer der Sigmoid Funktion nur die Identität unterstützt, die linear und nicht beschränkt ist, wird in dieser Arbeit ausschließlich die Sigmoid Funktion verwendet.

2.2.3 Training

Das Ergebnis der Klassifizierung hängt, abgesehen vom Input, zum großen Teil von den Gewichtungen ab. Es ist also essenziell, dass diese gut gewählt werden. Dafür gibt es verschiedene Algorithmen, von denen nun der „Backpropagation Algorithmus“ und der „Resilient Backpropagation Algorithmus“ (kurz „RPROP“), die beide in OpenCV implementiert sind, vorgestellt werden sollen.

Backpropagation Algorithmus

Die Idee des Backpropagation Algorithmus ist es, zunächst das Netzwerk mit zufälligen Gewichtungen zu belegen. Anschließend wird ein Inputvektor in das neuronale Netz eingespeist und der erhaltene Output mit dem gewünschten Output verglichen. Sind diese unterschiedlich, werden die Gewichtungen ein wenig angepasst, um den erhaltenen Output dem gewünschten Output anzunähern. Dieser Vorgang wird mit verschiedenen Inputvektoren wiederholt, bis man eine ausreichend genaue Klassifikation hat, beziehungsweise eine maximale Wiederholungsanzahl erreicht ist.
Die Gewichtungen von einer Schicht k zur jeweiligen nächsten Schicht $k+1$ werden (nach [5, S. 291]) folgendermaßen angepasst (η ist die konstante Lernrate und δ_i^k ist das „Fehlermaß“ des i-ten Neurons in der k-ten Schicht):

$$\Delta w_{ij}^k = -\eta \delta_i^{k+1} x_j^k$$

Die Berechnung des Fehlermaßes δ_i^k hängt von der Schicht k ab. Falls Schicht k die Outputschicht ist, gilt (mit t_i als dem gewünschten Output des Outputneurons x_i^k):

$$\delta_i^k = (t_i - x_i^k) f'(\sum_{j=0}^{d_k} x_j^k w_{ij}^k)$$

Andernfalls berechnet man δ_i^k folgendermaßen:

$$\delta_i^k = \sum_{j=1}^{d_{k+1}} w_{ji}^k \delta_j^{k+1} \cdot f'(\sum_{j=0}^{d_k} x_j^k w_{ij}^k)$$

Mit der Lernrate kann die Stärke bestimmt werden, mit der die bisherigen Gewichtungen verändert werden. Je größer sie ist, desto höher ist die Änderung an den Gewichtungen und umgekehrt.
Die neuen Gewichtungen können jetzt folgendermaßen berechnet werden:

$$w^k_{ij-neu} = w^k_{ij} + \Delta w^k_{ij}$$

RPROP-Algorithmus

Der RPROP Algorithmus funktioniert von der Idee her ähnlich wie der Backpropagation Algorithmus, allerdings hängt die Gewichtsänderung nicht von der Größe Δw^k_{ij}, sondern nur von dessen Vorzeichen ab (vgl. [10]). Sei $E^{k(t)}_{ij} = \delta^{k+1}_i x^k_j$ der Fehlerwert für die Gewichtung zwischen dem i-ten Neuron der k-ten Schicht und des j-ten Neuron der $(k+1)$-ten Schicht im t-ten Iterationsschritt. Weiter seien $\Delta^{k(t)}_{ij}$ Updatewerte im t-ten Iterationsschritt für die entsprechenden Gewichtungen w^k_{ij}. Nun seien $\Delta^{k(-1)}_{ij} = C$, wobei nach [10] $C = 0.1$ ein geeigneter Wert ist. Die Iterationsvorschrift des RPROP-Algorithmus ist nun (mit $0 < \eta^- < 1 < \eta^+$):

$$\Delta^{k(t)}_{ij} = \begin{cases} \eta^+ \cdot \Delta^{k(t-1)}_{ij} & \text{wenn } E^{k(t-1)}_{ij} \cdot E^{k(t)}_{ij} > 0 \\ \eta^- \cdot \Delta^{k(t-1)}_{ij} & \text{wenn } E^{k(t-1)}_{ij} \cdot E^{k(t)}_{ij} < 0 \\ \Delta^{k(t-1)}_{ij} & \text{sonst} \end{cases}$$

Also der Updatewert $\Delta^{k(t)}_{ij}$ nimmt zu, wenn das Vorzeichen von $E^{k(t)}_{ij}$ gleich bleibt und ab, wenn es sich ändert. Die auszuführende Gewichtungsänderung berechnet man nun folgendermaßen:

$$\Delta w^{k(t)}_{ij} = \begin{cases} -\Delta^{k(t)}_{ij} & \text{wenn } E^{k(t)}_{ij} > 0 \\ +\Delta^{k(t)}_{ij} & \text{wenn } E^{k(t)}_{ij} < 0 \\ 0 & \text{sonst} \end{cases}$$

Dadurch wird bei einem postiven Fehlerwert die Gewichtung verringert und anders herum. Man muss allerdings eine Ausnahme beachten:

$$\Delta w^{k(t)}_{ij} = -\Delta w^{k(t-1)}_{ij}, \text{ wenn } E^{k(t-1)}_{ij} \cdot E^{k(t)}_{ij} < 0$$

Da der Fehlerwert sein Vorzeichen geändert hat, wurde das Optimum im Schritt vorher übersprungen. Deswegen wird er rückgängig gemacht. Um zu vermeiden, dass der Updatewert im nächsten Schritt nochmal geändert wird, obwohl man die Gewichtung noch gar nicht geändert hat, setzt man $E^{k(t)}_{ij} := 0$. Dadurch bleibt im nächsten Schritt der Updatewert gleich.
Nun bleibt noch die Frage nach den Parametern η^+ und η^-. Laut [10] sind dafür (nach empirischer Überprüfung) die Werte $\eta^+ = 1,2$ und $\eta^- = 0,5$ am besten.

Kapitel 3

Bilderkennungsmerkmale

Im folgenden Kapitel geht es nun um drei verschiedene Bilderkennungsmerkmale, die im Hinblick auf die Fähigkeit Bildpunkte auf einer Straße von Bildpunkten außerhalb der Straße zu unterscheiden, untersucht werden sollen. Es werden dabei ausschließlich Graustufenbilder verwendet, das heißt es steht pro Pixel nur die Information „Helligkeit“ zur Verfügung. Es soll zunächst das Local Binary Pattern betrachtet werden, gefolgt von der diskreten Kosinustransformation und den HOG-Features.

3.1 Local Binary Pattern

Das Local Binary Pattern (LBP) stellt ein Maß für Grauwertänderungen an einem bestimmten Pixel zu seinen Nachbarn dar. Seine besondere Stärke liegt darin, dass es sehr tolerant gegenüber Helligkeitsänderungen ist (siehe [9]), da nur die lokalen Grauwertänderungen betrachtet werden. Eine Verdunkelung des Bildes (z. B. durch Schatten) nimmt darauf keinen Einfluss. Nun folgt die Vorstellung der Grundversion des LBP und dann wird dessen Verallgemeinerung erläutert.

3.1.1 Berechnung

Sei p_{ij} der Grauwert des Pixels eines Bildes an den Koordinaten i, j. Das Local Binary Pattern an der Stelle (i, j) wird dann folgendermaßen berechnet (für Indizes außerhalb des Bildes wird der Grauwert auf 0 gesetzt):

$$
\begin{aligned}
f(x) =& \begin{cases} 1 & \text{wenn } x \geq 0 \\ 0 & \text{wenn } x < 0 \end{cases} \\
s_{ij}(x, y) =& f(p_{(i+x)(j+y)} - p_{ij}) \\
LBP(i, j) =& 2^0 \cdot s_{ij}(-1, -1) + 2^1 \cdot s_{ij}(0, -1) + 2^2 \cdot s_{ij}(1, -1) + 2^3 \cdot s_{ij}(-1, 0) \\
& + 2^4 \cdot s_{ij}(1, 0) + 2^5 \cdot s_{ij}(-1, 1) + 2^6 \cdot s_{ij}(0, 1) + 2^7 \cdot s_{ij}(1, 1)
\end{aligned}
$$

Im Beispiel aus Abbildung 3.1 ergibt dies $4 + 8 + 64 + 128 = 204$.

1	1	6
7	5	4
2	5	8

0	0	1
1		0
0	1	1

Abbildung 3.1: Links: Die Zahlen in den Quadraten symboliesieren jeweils die Grauwerte, Rechts: Die Werte geben an, ob der Grauwert von der Mitte zu diesem Kästchen sinkt (0) oder gleich bleibt/steigt (1)

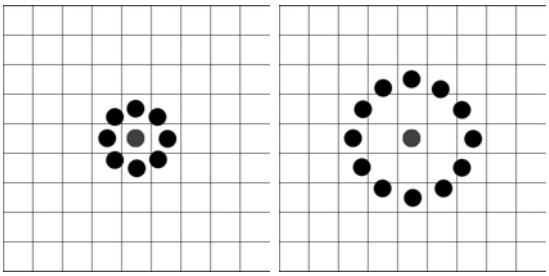

Abbildung 3.2: Nachbarschaftskreise für die Berechnung des LBP. Links die Grundversion mit R=1, P=8; Rechts mit R=2, P=12

3.1.2 Verallgemeinerung

Das Local Binary Pattern kann dahingehend verallgemeinert werden, dass man nicht mehr nur noch die Grauwertänderungen zu den acht benachbarten Pixeln betrachtet, sondern die zu einem „Nachbarschaftskreis“ mit einem bestimmten Radius R und Anzahl an Elementen P. Dies verdeutlicht Abbildung 3.2. Das LBP berechnet sich dann analog zur Grundversion durch die Darstellung der Grauwertänderungen als Binärzahl.

3.1.3 Verwendung als Texturmerkmal

Um das Local Binary Pattern nun als Texturmerkmal zur Klassifikation zu verwenden, bildet man Histogramme dieser Werte. Für das Histogramm an dem Punkt (i, j) wird ein Rechteck so um den Punkt gelegt, dass (i, j) sich in der Mitte befindet. Die Seitenlängen des Rechtecks sind dabei beliebig, sollten allerdings fest gesetzt werden, um eine bessere Vergleichbarkeit innerhalb einer Klassifikation zu wahren. Innerhalb dieses Rechtecks wird nun von allen Punkten das LBP berechnet und ein Histogramm der Werte gebildet. Dieses Histogramm wird dann als Datenvektor für die Klassifikation verwendet. Die Anzahl der Stellen des Datenvektors entspricht hier 2^P, bei der Grundversion 256.

3.2 HOG-Features

Nun soll ein weiteres Erkennungsmerkmal analysiert werden, nämlich die „Histogram of Oriented Gradients“ kurz HOG-Features. Die Idee hinter den HOG-Features ist, Objekte durch lokale Kanten- und Helligkeitsverläufe zu beschreiben, da sie dadurch oft gut erkannt werden können (siehe [4]).
Um die HOG-Features zu berechnen, werden zunächst die Gradienten in x und y-Richtung berechnet (nach [2]). Dafür filtert man das Bild mit den Kerneln $[-1, 0, 1]$ und $[-1, 0, 1]^T$, die Ergebnisse werden in I_x und I_y gespeichert. Daraus kann nun die Richtung Θ und die Norm m des Gradienten berechnet werden:

$$\Theta(x, y) = \tanh(\frac{I_y(x, y)}{I_x(x, y)})$$
$$m(x, y) = \sqrt{I_x(x, y)^2 + I_y(x, y)^2}$$

Nun wird das Bild in kleine Rechtecke eingeteilt, sogenannte Zellen, und für jede dieser Zellen ein Histogramm über die Gradientenrichtung erstellt. Dafür teilt man den Bereich von $0°$ bis $180°$ in n sogenannte Bins ein, man betrachtet dabei die Richtungen ohne Vorzeichen, da dies laut [4] bessere Ergebnisse liefert. Für $n = 9$ beispielsweise erhält man die Bins $0° - 20°$, …, $160° - 180°$. Anschließend addiert man jeweils von allen (x, y), deren Richtung $\Theta(x, y)$ in einem Bin liegt, die entsprechenden Gradientennorm $m(x, y)$ auf. Diese Summen bilden dann das Histogramm.
Dieses Vorgehen kann nun allerdings zu Problemen führen, da bei $n = 9$ z.B. ein $\Theta(x_1, y_1) = 19°$ genauso eingestuft wird, wie $\Theta(x_2, y_2) = 1°$, allerdings anders als $\Theta(x_3, y_3) = 21°$, obwohl (x_1, y_1) zu (x_3, y_3) viel ähnlicher ist. Ein anderes, ähnliches Problem ist, dass Gradienten, die fast genau auf der Grenze zwischen zwei Zellen liegen, nur in ein Histogramm eingehen, obwohl es schon durch kleine Messfehler zufällig scheint, in welcher der beiden Zellen sie landen.
Diese Probleme löst man dadurch, dass man die Norm nicht komplett zu einem Bin in einer Zelle hinzuaddiert, sondern anteilsmäßig auf die benachbarten Zellen und Bins verteilt, je nachdem wie nah diese sind. Die genaue Berechnung der Verteilung kann in [2, S. 14] nachgelesen werden.
Nun werden jeweils mehrere benachbarte Zellen zu sogenannten Blöcken zusammengefasst, in der Regel werden 2×2 Blöcke verwendet. Das Histogramm eines Blocks bildet man dann durch Hintereinanderreihung der Histogramme der einzelnen Zellen. Dieses Histogramm bildet nun den Featurevektor v eines Blocks. Anschließend wird v normiert, da sonst die Gradienten durch lokale Beleuchtungsänderungen verfälscht sein könnten. Dafür werden von [4] drei verschiedene Methoden empfohlen (für $\varepsilon > 0$ als kleine Konstante und mit $||(x_1, \ldots, x_n)||_1 = \sum_{i=1}^{n} |x_i|$):

- L_2-Norm: $v_{neu} = \frac{v}{\sqrt{||v||^2+\varepsilon^2}}$
- L_2-Hys: Wie $L_2 - Norm$, nur wird der Wertebereich auf $[0, \frac{1}{5}]$ eingeschränkt und anschließend wieder normiert.
- L_1-sqrt: $v_{neu} = \sqrt{\frac{v}{||v||_1+\varepsilon}}$

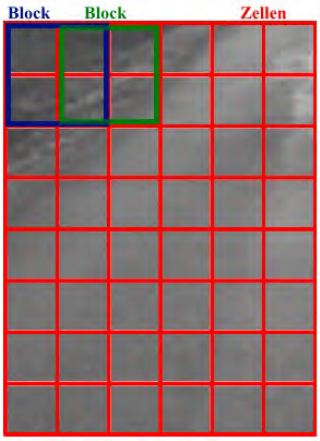

Abbildung 3.3: Aufteilung des Bildausschnitts in Zellen und 2×2 Blöcke

Die Blöcke werden hierbei jeweils überlappend gebildet, dies wird in Abbildung 3.3 verdeutlicht. Um jetzt den Featurevektor eines Bildausschnitts zu erhalten, reiht man alle Block-Featurevektoren hintereinander. Die Länge des Featurevektors ist also (mit $\#(x)$ als Anzahl der Objekte der Klasse x):

$$\#(\text{Blöcke}) \cdot \#(\text{Zellen pro Block}) \cdot \#(\text{bins})$$

3.3 Diskrete Kosinustransformation

Die diskrete Kosinustransformation wird in erster Linie zur Datenkompression genutzt. Allerdings findet sie auch in der Bilderkennung Anwendung (siehe [6]) und soll deswegen hier zur Fahrbahnerkennung getestet werden. Die Idee der DCT ist es, eine Folge an Inputwerten als Summe von Kosinusfunktionen darzustellen.
Die DCT $d(p)$ eines Datensatzes $p = (p_1, p_2, \ldots, p_n)$ berechnet sich durch (nach [1]):

$$d_0 = \frac{\sqrt{2}}{n} \sum_{i=0}^{n-1} p(n)$$

$$\forall k \in \{1, \ldots, n-1\} : d_k(p) = \frac{2}{n} \sum_{i=0}^{n-1} p(n) \cos\left(\frac{(2i+1)\pi k}{2n}\right)$$

Den Datensatz p erhält man dadurch, dass man die Grauwerte eines rechteckigen Bildausschnittes aneinander reiht. Die erhaltenen Werte $d(p)$ bilden dann den Featurevektor des Mittelpunktes des Rechtecks. Die Länge des Featurevektors entspricht hier dem Produkt aus Länge und Breite des Bildausschnittes.

Kapitel 4

Test der vorgestellten Methoden

In den vorherigen Kapiteln wurden zwei Klassifikatoren und drei Texturmerkmale vorgestellt. Jetzt sollen diese getestet, beziehungsweise ihre verschiedenen Variationsmöglichkeiten ausgewertet und verglichen werden. Anschließend werden noch weitere Experimente zur Verbesserung der Fahrbahnerkennung vorgestellt.
Um die Tests durchführen zu können wurde zu den verwendeten Videos händisch eingegeben, in welchem Bereich sich auf den jeweiligen Einzelbildern Fahrbahn befindet. Diese Informationen liest das Testprogramm ein, berechnet anhand des gewählten Bilderkennungsmerkmals Testdaten auf und neben der Fahrbahn und klassifiziert diese wie vorgegeben. Anschließend können Klassifikatoren mit diesen Daten trainiert werden oder es kann eine Cross-Validation mit diesen Daten durchgeführt werden.
Zum Testen der Klassifikatoren wurden zwei verschiedenen Strecken A und B (eine, bei der das Training erfahrungsgemäß gut funktioniert und eine bei der es nicht so gut klappt) verwendet. Im Anhang finden sich Bilder aus den Strecken (Abbildung 5.1 und Abbildung 5.2). Zu diesen beiden Strecken wurden jeweils einmal mit dem Local Binary Pattern und einmal mit der diskreten Kosinustransformation die Testdaten berechnet und damit die Klassifikatoren anhand der Cross-Validation evaluiert. Als Rechteckgröße zur Merkmalsberechnung wurde dabei 40×40 verwendet.

4.1 Test der Support Vektor Machines

Nun sollen die Support Vektor Machines untersucht werden und dabei die verschiedenen Kernelfunktionen verglichen werden, die in Kapitel 2.1.5 vorgestellt wurden.

4.1.1 Lineare Kernelfunktion

Angewandt auf die vier verschiedenen Test-Konfigurationen erreicht die lineare Kernelfunktion folgenden (gerundeten) F_1-Werte: 0,733; 0,975, 0,566; 0,745.
Der gerundete Durchschnitt dieser Werte ist 0,755. Dies entspricht einer sehr niedrigen Genauigkeit, vor allem wenn man bedenkt, das bereits durch reines Raten im Durschschnitt der Wert 0,5 erreicht wird. Die lineare Kernelfunktion sollte in diesem Anwendungsbereich also eher außen vor gelassen werden, da sie nicht die scheinbar nötige Komplexität besitzt.

4.1.2 Polynomiale Kernelfunktion

Die polynomiale Kernelfunktion hängt von den Parametern γ, c_0 und d ab. Um gute Parameter zu bestimmen, wird hierbei für das Training die Funktion „train_auto“ aus der C-Bibliothek „OpenCV“ verwendet, die eine automatische Parametersuche mit dem „Grid-Search“Verfahren durchführt und möglichst optimale Parameter bestimmt.
Mit den vier verschiedenen Test-Konfigurationen erreicht diese Kernelfunktion folgende (gerundete) F_1-Werte: 0,758; 0,932; 0,893; 0,982.
Der Durchschnitt ist 0,891 und damit ist das Ergebnis deutlich besser als bei der linearen Kernelfunktion. Allerdings steigt die Dauer des Trainings durch die Funktion „train_auto“ stark an, da die Suche der Parameter sehr aufwändig ist. Da das Training aber nur bei der Initialisierung der Support Vector Machine durchgeführt werden muss, wäre die längere Trainingslaufzeit kein Nachteil beim Einsatz im Straßenverkehr.

4.1.3 Radial Basis Kernelfunktion

Hier wird erneut ein Parameter für die Kernelfunktion benötigt, nämlich γ. Es wird wieder die Funktion „train_auto“ verwendet um einen guten Parameter zu erhalten.
Diese Kernelfunktion erreicht, auf die Test-Konfigurationen angewedet, folgende (gerundete) F_1 Werte: 0,827; 0,941; 0,968; 0,981.
Der Durchschnitt ist hier 0,929, also noch einmal besser als bei der polynomialen Kernelfunktion. Auch zeitlich schneidet dieser Kernel besser ab, da er zwei Parameter weniger hat und somit zwei Parameter weniger durch „train_auto“ optimiert werden müssen, was den größten Zeitanteil einnimmt.

4.1.4 Sigmoid Kernelfunktion

Auch hier wird wieder die Funktion „train_auto“ verwendet um gute Parameter γ und c_0 zu erhalten.
Diese Kernelfunktion scheint jedoch nicht für diese Anwendung geeignet zu sein. Nur eine der vier Testkonfigurationen (jeweils die zweite in den vorhergehenden Tests) liefert ein gutes Ergebniss (F_1-Wert von 0,901), zwei andere liefern 0,486 und 0,624 und bei der letzten werden einfach alle Punkte als Straße klassifiziert.

4.1.5 Fazit

Zusammenfassend lässt sich sagen, dass die Radial Basis Kernelfunktion vermutlich die beste Klassifikation von Fahrbahnen liefert. Das deckt sich auch mit der Empfehlung aus der OpenCV-Dokumentation, dass diese in den meisten Fällen eine gute Wahl ist. Am zweitbesten schneidet die polynomiale Kernelfunktion ab, diese ist allerdings zusätzlich auch noch weniger perfomant.

4.2 Test der neuronalen Netze

Nun sollen die neuronalen Netze getestet werden, die in Kapitel 2.2 erklärt werden. Dabei wird zunächst ein Vergleich der beiden Trainingsalgorithmen durchgeführt, anschließend

Anzahl Schichten	Anzahl Neuronen	Durchschnitt F_1 RPROP	Durchschnitt F_1 Backpropagation
2	-	0,931	0,936
3	2	0,933	0,861
3	5	0,932	0,933
3	30	0,919	0,942
3	100	0,903	0,934
4	2,2	0,933	0,731
4	20,20	0,936	0,919
4	100,10	0,902	0,867

Tabelle 4.1: Übersicht über die F_1-Werte verschiedener Netzstrukturen.

wird zusätzlich auf die Netzwerkstruktur eingegangen. Schließlich wird noch überprüft wie hoch die Abweichung von Tests bei gleichem Testaufbau sein kann.

4.2.1 Test der Trainingsmethoden

Im diesem Abschnitt wird empirisch überprüft, welche der beiden im Kapitel 2.2.3 vorgestellten Trainingsalgorithmen im Bezug auf Fahrbahnerkennung besser funktioniert. Es wird zunächst ein dreischichtiges Neuronales Netz mit zwei Neuronen im Hidden Layer für die Klassifikation verwendet.
Der Backpropagation Algorithmus liefert folgende vier F_1-Werte: 0,824; 0,863; 0,911; 0,846, das ist im Durchschnitt gerundet 0,861. Der RPROP Algorithmus liefert deutlich bessere Werte, nämlich 0,834; 0,95; 0,978; 0,969 und erreicht damit im Schnitt 0,933.
Bei einem dreischichtigen Netz mit 100 Neuronen im Hidden Layer ergibt sich für den Backpropagation Algorithmus allerdings ein Durchschnitt von 0,934, für den RPROP Algorithmus nur ein Durchschnitt von 0,903. Da die optimale Trainingsmethode also eventuell auch von der Netzwerkstruktur abhängig ist, wird der Test nun dementsprechend erweitert.

4.2.2 Test verschiedener Netzwerkstrukturen

Jetzt wird zusätzlich getestet, wie sich die Netzwerkstruktur auf die Testergebnisse auswirkt. Eine Übersicht über die Ergebnisse ist in Tabelle 4.1 dargestellt. Die Anzahl der Neuronen in der zweiten Spalte bezieht sich dabeil jeweils nur auf die Hidden Layers, da Input und Output Layer fest sind.
Insgesamt lässt sich sagen, dass keine klare Tendenz zu erkennen ist, ob kleinere oder größere Netzwerke besser klassifizieren. Da sich kleinere Netzwerke schneller trainieren und auswerten lassen, sollten also diese bevorzugt werden. Von den Trainingsalgorithmen schneidet der RPROP Algorithmus besser ab, da er weniger negative Ausreißer hat. Eine Tabelle mit noch ausführlicheren Tests, die diese Schlussfolgerungen bestätigen gibt es im Anhang (Tabelle 5.1).

4.2.3 Abweichung der Tests

Da die Gewichtungen am Anfang des Trainings mit zufälligen Werten belegt werden, können sich die aus dem Training erhaltenen Klassifikatoren trotz gleicher Trainingsdaten unterscheiden. Um die Aussagekraft einzelner Testergebnisse einschätzen zu können, soll nun getestet werden, wie weit einzelne Ergebnisse, abhängig von der Anzahl an Trainingsdaten, voneinander abweichen können. Eine Übersicht über die Ergebnisse ist in Tabelle 4.2 zu sehen, die Berechnung der Standardabweichung beruht dabei auf einer Stichprobe von 50 Werten.

Anzahl	Testergebnisse	Standardabweichung
655	0,969 0,969 0,968 0,970 ...	0,00466
5.861	0,955 0,950 0,954 0,949 ...	0,00636
25.816	0,949 0,942 0,952 0,945 ...	0,00810
1.153	0,868 0,867 0,874 0,871 ...	0,00948
9.126	0,811 0,809 0,807 0,812 ...	0,00409
39.214	0,807 0,807 0,810 0,795 ...	0,00288

Tabelle 4.2: Verschiedene F_1 Werte für ein neuronales Netz bei der Verwendung der gleichen Trainingdaten

4.3 Test der Bilderkennungsmerkmale

Für die Versuche mit den Bilderkennungsmerkmale wurden (jeweils mit dem entsprechenden Merkmal) wieder zu den beiden Strecken A und B die Trainingsdaten berechnet. Anschließend wurden sowohl eine Support Vektor Machine als auch ein neuronales Netz mit diesen Trainingsdaten und mithilfe der Cross-Validation ausgewertet und der Durschnitt der vier F_1-Werte gebildet.

4.3.1 Test des LBP

Jetzt folgen Tests zu dem in Kapitel 3.1 vorgestellten Local Binary Pattern. Dabei sollen verschiedenen LBPs mit verschiedenen Seitenlängen des Histogramm-Rechtecks getestet werden und die Ergebnisse analysiert werden. Eine Übersicht über die Testergebnisse ist in Tabelle 4.3 zu sehen.

R	P	Durchschnitt F_1 20x20	Durchschnitt F_1 20x40	Durchschnitt F_1 40x20	Durchschnitt F_1 40x40
1	8	0,906	0,936	0,931	0,956
2	8	0,892	0,931	0,940	0,952
2	12	0,812	0,839	0,826	0,849
2,5	12	0,781	0,831	0,819	0,858

Tabelle 4.3: Übersicht über die F_1-Werte verschiedener LBPs mit verschiedenen Größen (Breite x Höhe) für das Histogramm-Rechteck

Zusammenfassend lässt sich sagen, dass das 40×40 Rechteck am besten abschneidet. Außerdem erreichen die LBPs mit P=8 eine klar bessere Klassifizierung als die LBPs mit P=12. Weitere Testergebnisse gibt es im Anhang in Tabelle 5.2.

4.3.2 Test der HOG-Features

Jetzt sollen die in Kapitel 3.2 beschriebenen HOG-Features getestet werden. Dazu werden ausschließlich 4×4 Zellen mit 9 Bins und 2×2 Blöcke verwendet. Die Größe des Bildausschnitts variiert je nach Testdurchlauf, die Ergebnisse sind in Tabelle 4.4 zu sehen. Die Testergebnisse lassen vermuten, dass größere Bildauschnitte, wie bei den LBPs auch, etwas besser abschneiden. Eine umfangreiche Tabelle, die diese Vermutung bestätigt, ist im Anhang zu finden (Tabelle 5.3). Größere Bildausschnitte haben allerdings den Nachteil, dass es mehr Bildsegmente gibt, auf denen teilweise Straße zu sehen ist und teilweise nicht (dieser Nachteil schlägt sich jedoch nicht in den Testergebnissen wieder, da diese Bildausschnitte nicht in die Testdaten eingehen). Das führt dazu, dass die Straße in der Mitte besser erkannt werden kann, allerdings der genaue Fahrbahnrand eventuell nicht.

Breite	Höhe	Durchschnitt F_1
16	16	0,867
16	32	0,888
32	16	0,904
32	32	0,899
32	48	0,899
48	32	0,891
48	48	0,915

Tabelle 4.4: Übersicht über die F_1-Werte verschiedener HOG-Features mit verschiedenen Größen für den Bildauschnitt

4.3.3 Test der DCT

Nun sollen verschiedene Rechteckgrößen für die DCT getestet werden, die in Kapitel 3.3 erläutert wurden. Einen Überblick über die Testergebnisse ist in Tabelle 4.5 zu sehen.
In diesem Fall schneiden eher die kleineren Rechtecke besser ab, viel kleiner als 10×10 sollten sie allerdings nicht werden. Auch hier finden sich im Anhang in Tabelle 5.4 ausführliche Testergebnisse.

4.4 Weitere Experimente

Nun sollen einige weitere Versuche beschrieben werden, wie man die Fahrbahnerkennung noch weiter verbessern könnte. Dabei wird zunächst untersucht, wie die simple Verwendung der Bildgrauwerte als Datenvektor abschneidet. Anschließend werden verschiedene Texturmerkmale kombiniert und das Ergebnis der Klassifikation untersucht.

Breite	Höhe	Durchschnitt F_1
10	10	0.862
10	20	0.865
20	10	0.864
20	20	0,859
20	40	0,848
40	20	0,836
40	40	0,831
40	60	0,821

Tabelle 4.5: Übersicht über die F_1-Werte verschiedener Rechteckgrößen der DCT

4.4.1 Grauwerte als Merkmal

Bei diesem Bilderkennungsmerkmal werden einfach die Grauwerte eines Rechtecks um einen bestimmten Pixel herum als Featurevektor des Pixels verwendet. Die Länge des Featurevektors entspricht also der Länge mal der Breite des Rechtecks. Diese Methode verspricht allerdings nicht sehr erfolgreich zu sein, da sie in bisherigen Arbeiten zu diesem Thema kaum erwähnt wird, trotzdem soll sie hier getestet werden.
Wie Tabelle 4.6 zeigt, können die Grauwerte als Merkmal wie erwartet nicht mit den vorher gezeigten Texturmerkmalen mithalten. Der einzige Vorteil bleibt die Schnelligkeit, da keine Berechnung notwendig ist.

Breite	Höhe	Durchschnitt F_1
8	8	0,834
8	16	0,823
16	8	0,846
16	16	0,834
16	32	0,793
32	16	0,822
32	32	0,843

Tabelle 4.6: Übersicht über die F_1-Werte verschiedener Rechteckgrößen mit dem Grauwertmerkmal

4.4.2 Kombination verschiedener Texturmerkmale

Ein Versuch, die Fahrbahnerkennung weiter zu verbessern, ist die Kombination von Texturmerkmalen. Durch die Aneinanderreihung der Featurevektoren der einzelnen Merkmale stehen dem Klassifikator mehr Informationen zur Verfügung, wodurch er, auf Kosten der Geschwindigkeit, eventuell eine bessere Klassifikation findet.

LBP und HOG

Begonnen wird mit dem Test der Kombination des Local Binary Patterns und der HOG-Features. Diese Kombination wird außerdem in [13] empfohlen.

Breite	Höhe	Durchschnitt F_1
16	16	0,914
16	32	0,929
32	16	0,931
32	32	0,938
32	48	0,941
48	32	0,947
48	48	0,945

Tabelle 4.7: Übersicht über die F_1-Werte der Kombination LBP und HOG

Die Ergebnisse, die in Tabelle 4.7 präsentiert werden, weisen allerdings keine Verbesserung im Vergleich zur einfachen Verwendung des LBPs auf. Aufgrund des erhöhten Rechenaufwandes ist also von dieser Kombination abzusehen. Ausführliche Testergebnisse sind in Tabelle 5.5 zu sehen.

LBP und DCT

Nun soll die Kombination des LBP mit der DCT untersucht werden. Die Ergebnisse sind in Tabelle 4.8 dargestellt.
Das LBP und die DCT schneiden zusammen erkennbar schlechter ab als das LBP alleine, zusätzlich ist ein erhöhter Rechenaufwand nötig. Auch hier ist also die Klassifikation trotz zusätzlicher Informationen schlechter, das kann daran liegen, dass der Klassifikator durch die zusätzlichen Informationen die entscheidenden Informationen nicht mehr so leicht erkennt.

Breite	Höhe	Durchschnitt F_1
16	16	0,871
16	32	0,902
32	16	0,912
32	32	0,910
32	48	0,902
48	32	0,922
48	48	0,919

Tabelle 4.8: Übersicht über die F_1-Werte der Kombination LBP und DCT

HOG und DCT

Schließlich soll noch die letzte verbleibende Kombination, HOG-Features und die DCT, getestet werden.
Die in Tabelle 4.9 zu sehenden Ergebnisse sind minimal besser als die der HOG-Features (oder auch der DCT) alleine, allerdings immer noch bei weitem nicht so gut wie die des LBPs.

Breite	Höhe	Durchschnitt F_1
16	16	0,894
16	32	0,898
32	16	0,911
32	32	0,896
32	48	0,912
48	32	0,910
48	48	0,906

Tabelle 4.9: Übersicht über die F_1-Werte der Kombination HOG und DCT

4.5 Fazit

In den vorherigen Abschnitten wurden die beiden Klassifikatoren und die Texturmerkmale ausführlich getestet. Während die beiden Klassifikatoren in der Spitze ähnlich gut abschneiden (F_1-Werte von ungefähr 0,93), gibt es bei den Bilderkennungsmerkmalen erkennbare Unterschiede. Das Local Binary Pattern erreicht mit ca. 0,95 bei einer Rechteckgröße von 40×40 den besten F_1-Wert, während die HOG-Features und die DCT mit 0,90 bzw. 0,86 klar schlechter abschneiden.
Die Verwendung der Grauwerte liefert, wie erwartet, auch keine besseren Ergebnisse. Selbst die Kombination verschiedener Texturmerkmale kommt, obwohl es die meisten Informationen zur Verfügung hat, nicht an das LBP heran.

Kapitel 5

Ausblick

Im vorherigen Kapitel wurden die in dieser Arbeit behandelten Methoden zur Fahrbahnerkennung getestet. Bei geigneter Wahl eines Bilderkennungsmerkmals können mit ihnen F_1-Werte von über 0,95 erreicht werden. Allerdings sind die Sichherheitsanforderungen im Straßenverkehr sehr hoch, weshalb nach weiteren Methoden zur Verbesserung gesucht werden sollte.
Einige Möglichkeiten dazu würden sich eventuell aus der Verwendung von Farbbildern anstatt der Graustufenbilder ergeben, da dann mehr Informationen pro Pixel zur Verfügung stehen. Es gibt bereits viele unterschiedliche Ansätze, die sich mit der Segmentierung von Farbbildern beschäftigen. Die meisten davon verwenden schon aus der Graustufenbildanalyse bekannte Bildmerkmale einfach auf allen Farbkanälen und kombinieren diese. Eine Übersicht über einige verbreitete Methoden ist in [7] zu finden.
Auch wenn im Bereich der Bilderkennung schon viel geforscht wurde, es gibt immer noch viel zu tun.

Danksagung

Ich danke Florian Janda für die Unterstützung während des Erstellens dieser Arbeit und für das Korrekturlesen selbiger. Außerdem möchte ich ihm für die Implementierung der DCT und allen sonstigen Hilfestellungen bei Problemen während der Entwicklung des Testprogramms danken.
Weiterhin danke ich Herrn Prof. Dr. Sauer für die Möglichkeit eine Bachelorarbeit zu diesem Thema zu verfassen.

Erklärung

Hiermit versichere ich, dass ich diese Bachelorarbeit selbstständig und ohne Benutzung anderer als der angegebenen Quellen und Hilfsmittel angefertigt habe und alle Ausführungen, die wörtlich oder sinngemäß übernommen wurden, als solche gekennzeichnet sind, sowie dass ich diese Bachelorarbeit in gleicher oder ähnlicher Form noch keiner anderen Prüfungsbehörde vorgelegt habe.

Passau, 14. Oktober 2014

Literatur

[1] N. Ahmed, T. Natarajan und K. R. Rao. "Discrete Cosine Transform". In: *IEEE Transactions on Computers* (Januar, 1973). URL: http://web.iaincirebon.ac.id/ebook/luke/ieeeexplore/Computers_IEEE_Transactions_o/Discrete_Cosine_Transfom-0yr.pdf.

[2] Marco Blauth. *Detektion und Klassifikation von Verkehrszeichen*. 2012. URL: http://www.itwm.fraunhofer.de/fileadmin/ITWM-Media/Abteilungen/BV/Pdf/Bachelorarbeit_Marco_Blauth.pdf.

[3] Chih-Chung Chang und Chih-Jen Lin. *LIBSVM: A Library for Support Vector Machines*. Techn. Ber. National Taiwan University, Taipei, Taiwan, 2013. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf.

[4] Navneet Dalal und Bill Triggs. *Histograms of Oriented Gradients for Human Detection*. Techn. Ber. INRIA Rhone-Alps, 2005. URL: http://hal.archives-ouvertes.fr/docs/00/54/85/12/PDF/hog_cvpr2005.pdf.

[5] Richard O. Duda, Peter E. Hart und David G. Stork. *Pattern Classification*. 2nd. New York: John Wiley & Sons, Inc., 2001.

[6] Ziad M. Hafed und Martin D. Levine. "Face Recognition Using the Discrete Cosine Transform". In: *International Journal of Computer Vision* (2001). URL: http://www.physiol-active-vision.uni-tuebingen.de/paper/fr_IJCV_2001.pdf.

[7] Dana E. Ilea und Paul F. Whelan. "Image segmentation based on the integration of colour–texture descriptors—A review". In: *Pattern Recognition Volume 44, Issues 10–11* (2011). URL: http://www.sciencedirect.com/science/article/pii/S0031320311000902.

[8] Bernd Kuhlenschmidt. *Support Vector Machines*. Techn. Ber. Universität Münster, 2008. URL: http://www.math.uni-muenster.de/u/lammers/EDU/ws07/Softcomputing/Abgaben/%5B2A%5D%20-%20SupportVectorMachines_Final.pdf.

[9] Matti Pietikäinen. "Image Analysis". In: Springer, Berlin Heidelberg, 2005. Kap. Image Analysis with Local Binary Patterns. URL: http://www.ee.oulu.fi/mvg/files/pdf/pdf_638.pdf?origin=publication_detail.

[10] Martin Riedmiller und Heinrich Braun. *A Direct Adaptive Method for Faster Backpropagation Learning: The RPROP Algorithm*. Techn. Ber. Institut fur Logik, Komplexitat und Deduktionssyteme, University of Karlsruhe, 1993. URL: http://www.csie.ntu.edu.tw/~cjlin/papers/libsvm.pdf.

[11] opencv dev team. *OpenCV Documentation zu ihrer Implementatierung der Support Vector Machine*. Stand 12.08.2014. Apr. 2014. URL: `http://docs.opencv.org/modules/ml/doc/support_vector_machines.html`.

[12] opencv dev team. *OpenCV Documentation zu ihrer Implementatierung des Neuronalen Netzes*. Stand 12.08.2014. Apr. 2014. URL: `http://docs.opencv.org/modules/ml/doc/neuronal_network.html`.

[13] Xiaoyu Wang, Tony X. Han und Shuicheng Yan. "An HOG-LBP human detector with partial occlusion handling". In: *2009 IEEE 12th International Conference on Computer Vision* (2009). URL: `http://www.cs.ucf.edu/courses/cap6412/fall2009/papers/Wang_Han_Yan_iccv09.pdf`.

Anhang

Tabelle 5.1

Übersicht über die F_1-Werte vieler verschiedener Netzstrukturen. Die Anzahl der Neuronen bezieht sich jeweils nur auf die hidden layers, da input und output layer fest sind.

Anzahl Schichten	Anzahl Neuronen	Durchschnitt F_1 RPROP	Durchschnitt F_1 Backpropagation
2	-	0,931	0,936
3	2	0,933	0,861
3	3	0,936	0,936
3	5	0,932	0,933
3	10	0,930	0,934
3	30	0,919	0,942
3	50	0,912	0,933
3	100	0,903	0,934
4	2,2	0,933	0,731
4	20,2	0,938	0,869
4	20,10	0,932	0,909
4	20,20	0,936	0,919
4	50,5	0,929	0,914
4	50,10	0,931	0,928
4	100,10	0,902	0,867
5	100,30,5	0,928	0,844

Tabelle 5.2

Übersicht über die F_1-Werte verschiedener LBPs mit verschiedenen Größen (Breite, Höhe) für das Histogramm-Rechteck

R	P	Breite	Höhe	Durcschnitt F_1
1	8	20	20	0,906
1	8	20	30	0,921
1	8	20	40	0,936
1	8	30	20	0,926
1	8	30	30	0,943
1	8	30	40	0,950
1	8	40	20	0,931
1	8	40	30	0,946
1	8	40	40	0,956
1	8	40	50	0,960
2	8	20	20	0,892
2	8	20	30	0,910
2	8	20	40	0,931
2	8	30	20	0,923
2	8	30	30	0,934
2	8	30	40	0,941
2	8	40	20	0,940
2	8	40	30	0,945
2	8	40	40	0,952
2	8	40	50	0,962
2	10	20	20	0,858
2	10	20	30	0,870
2	10	20	40	0,894
2	10	30	20	0,878
2	10	30	30	0,900
2	10	30	40	0,912
2	10	40	20	0,888
2	10	40	30	0,908
2	10	40	40	0,926
2	10	40	50	0,914
2	12	20	20	0,812
2	12	20	30	0,820
2	12	20	40	0,839
2	12	30	20	0,824
2	12	30	30	0,802
2	12	30	40	0,831
2	12	40	20	0,826
2	12	40	30	0,842
2	12	40	40	0,849

2	12	40	50	0,862
2,5	12	20	20	0,781
2,5	12	20	30	0,811
2,5	12	20	40	0,831
2,5	12	30	20	0,821
2,5	12	30	30	0,832
2,5	12	30	40	0,843
2,5	12	40	20	0,819
2,5	12	40	30	0,840
2,5	12	40	40	0,858
2,5	12	40	50	0,878

Tabelle 5.3

Übersicht über die F_1-Werte verschiedener HOG-Features mit verschiedenen Größen für den Bildauschnitt.

Breite	Höhe	Durchschnitt F_1
16	16	0,867
16	24	0,877
16	32	0,888
24	16	0,886
24	24	0,887
24	32	0,884
24	40	0,898
24	48	0,890
32	16	0,904
32	24	0,884
32	32	0,899
32	40	0,908
32	48	0,899
32	56	0,903
40	24	0,897
40	32	0,893
40	40	0,908
40	48	0,914
40	56	0,910
48	24	0,908
48	32	0,891
48	40	0,913
48	48	0,915
48	56	0,909
48	64	0,919

Tabelle 5.4

Übersicht über die F_1-Werte verschiedener Rechteckgrößen der DCT.

Breite	Höhe	Durchschnitt F_1
4	4	0,824
4	6	0,838
6	4	0,834
6	6	0,850
6	10	0,861
10	6	0,793
10	10	0.862
10	16	0,869
10	20	0.865
16	10	0,871
16	16	0,858
16	20	0,870
16	30	0,857
20	10	0.864
20	16	0,859
20	20	0,859
20	30	0,818
20	40	0,848
30	16	0,872
30	20	0,872
30	30	0,833
30	40	0,860
30	50	0,844
40	20	0,836
40	30	0,849
40	40	0,831
40	50	0,856
40	60	0,821

Tabelle 5.5

Übersicht über die F_1-Werte verschiedener Rechteckgrößen der Kombination aus den HOG-Features und dem LBP.

Breite	Höhe	Durchschnitt F_1
8	8	0,88
8	16	0,882
16	8	0,899
16	16	0,914
16	24	0,918
16	32	0,929
24	16	0,926
24	24	0,925
24	32	0,929
24	40	0,931
24	48	0,943
32	16	0,931
32	32	0,938
32	40	0,942
32	48	0,941
40	24	0,937
40	32	0,947
40	40	0,943
40	48	0,947
48	24	0,940
48	32	0,947
48	40	0,945
48	48	0,945

Tabelle 5.6

Übersicht über die F_1-Werte verschiedener Rechteckgrößen der Kombination aus dem LBP und der DCT.

Breite	Höhe	Durchschnitt F_1
8	8	0,876
8	16	0,898
16	8	0,892
16	16	0,871
16	24	0,886
16	32	0,902
24	16	0,903
24	24	0,913
24	32	0,909
24	40	0,912
24	48	0,909
32	16	0,912
32	24	0,914
32	32	0,910
32	40	0,899
32	48	0,902
40	24	0,924
40	32	0,897
40	40	0,907
40	48	0,919
48	24	0,922
48	32	0,922
48	40	0,925
48	48	0,919

Tabelle 5.7

Übersicht über die F_1-Werte verschiedener Rechteckgrößen der Kombination aus den HOG-Features und der DCT.

Breite	Höhe	Durchschnitt F_1
8	8	0,87
8	16	0,877
16	8	0,89
16	16	0,894
16	24	0,892
16	32	0,898
24	16	0,903
24	24	0,901
24	32	0,874
24	40	0,895
24	48	0,900
32	16	0,911
32	24	0,907
32	32	0,896
32	40	0,883
32	48	0,912
40	24	0,917
40	32	0,908
40	40	0,907
40	48	0,902
48	24	0,907
48	32	0,910
48	40	0,912
48	48	0,906

Abbildung 5.1

Zwei Bilder aus dem Video zu Strecke A, die weißen Rechtecke represäntieren als „Straße“ eingelesene Bildausschnitte, die schwarzen Rechtecke stehen für „nicht Straße“.

Abbildung 5.1: Zwei Bilder aus dem Video zu Strecke A

Abbildung 5.2

Zwei Bilder aus dem Video zu Strecke B, die weißen Rechtecke represäntieren als „Straße“ eingelesene Bildausschnitte, die schwarzen Rechtecke stehen für „nicht Straße“.

Abbildung 5.2: Zwei Bilder aus dem Video zu Strecke B