# Inhaltsverzeichnis

# Schrifterkennung

## mit SVMs und Neuronalen Netzen

## 1 Einleitung

Täglich werden in Deutschland rund 64 Millionen Briefe verschickt. Doch habt ihr euch schon einmal gefragt, wie die ganzen Briefe so schnell verarbeitet werden können? Die Vorstellung von mehreren tausend Mitarbeitern, die sich nur mit der Sortierung der Post beschäftigen, erscheint schon aufgrund des Aufwandes unwahrscheinich. Eine bessere Variante stellt die Benutzung von Schrifterkennungsalgorithmen dar, deren sich die Post seit längerer Zeit bedient. Dabei scannt ein Computer die auf den Umschlägen geschriebenen Adressen ein, wertet diese aus und sortiert sie entsprechend.
Zu diesem Thema gab es zwei Projektgruppen, die sich dem Problem auf verschiedene Weise näherten. Eine Gruppe nutzte SVMs, die andere ein neuronales Netzwerk.

*Abb. 1.1: Schwarz-Weiß Bild eines Kameramannes.*

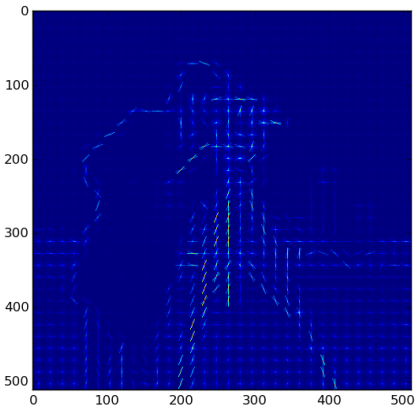


*Abb. 1.2: Feature-Bild des voherigen Bildes.*

## 2 Schrifterkennung mit SVMs

**Aufbau des Programms** Da es beim maschinellen Lernen essenziell ist über ein entsprechend großes Datenset zu verfügen, haben wir unseren Algorithmus zunächst einmal mit 60.000 Bildern ausgestattet.

```
feature, hog_image = feats.hog(sub_pic,
 orientations = 8,pixels_per_cell = (4,4),
 cells_per_block = (1,1),visualise = true)
```

Im ersten Schritt generieren wir die Features sämtlicher Bilder. Features sind Muster in einem Bild, z.B. Kanten oder Farben. Dabei gibt es mehrere Einstellungsmöglichkeiten, beispielsweise zur Variation der Pixelanzahl in einer Zelle (pixels_per_cell) und der Genauigkeit zur Erkennung von Konturen (orientations). Dabei gilt: Je kleiner die Zellen sind und je höher die Genauigkeit ist, desto länger braucht der Algorithmus zum berechnen, aber desto präziser ist das Endergebnis. Allerdings muss man auch beachten, dass die Zellengröße nicht zu klein wird, da in sehr kleinen Zellen die Muster eventuell nicht für das Programm erkennbar sind.

```
clf[:fit](features_trainset',
 label_train_data[:])
 ...
prob = clf[:predict_proba]
 (features_testset')
```

Anschließend wird die SVM mit den Bildern trainiert und ist somit einsatzbereit.Um nun die Genauigkeit der SVM zu testen, erzeugen wir ein Testset, dessen Features ebenfalls über den bestehenden Algorithmus generiert werden. Getestet wird die SVM, indem wir das Testset an die SVM übergeben und sie uns aufgrund der vorhandenen Werte eine Schätzung abgibt, was am wahrscheinlichsten auf dem Bild abgebildet ist. Die Ergebnisse werden dann überprüft und die Genauigkeit wird errechnet.

Daraufhin erzeugen wir ein Bild mit 100 Zahlen Abb. 0.0, die jeweils in einer $38 \times 38$ Pixelzelle zufällig verteilt werden. Dieses Bild wird in eine Matrix umgewandelt und ein Algorithmus iteriert

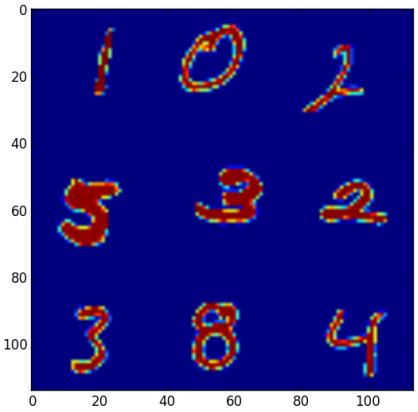

*Abb. 1.3: Ausschnitt des generierten Bildes, das die SVM analysieren soll.*

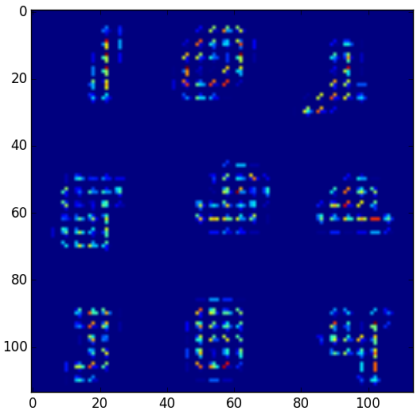

*Abb. 1.4: Feature-Bild des generierten Bildes.*

über die Matrix, sodass die SVM bestimmen kann, welche 100 Zahlen abgebildet sind.

# 3 Zeichenerkennung mit neuronalen Netzen

Ein funktionsfähiges neuronales Netz zu implementieren, bedeutet einen großen Zeitaufwand und viel Feinoptimierung. Da uns diese Zeit nicht zur Verfügung stand und wir ein größeres Interesse an der Anwendung hatten, nutzten wir Caffe. Bei Caffe handelt es sich um eine Software des Berkeley Vision and Learning Centers. Caffe bietet eine Art Bausatz für neuronale Netze. Man kann verschiedene Filter auswählen und sich so sein eigenes neuronales Netz erstellen. Caffe ist in der Lage Bilder sehr schnell auszuwerten, nach Herstellerangaben benötigt es je Bild 4 ms in der Trainingsphase und 1 ms in der Testphase.
Die für die Zeichenerkennung notwendigen Daten beziehen wir aus der MNIST-Datenbank, die 60.000 handgeschriebene Beispiele für Zahlen und 10.000 Beispiele für Test-Daten enthält. Diese vorgefertigte Zeichendatenbank eignet sich vor allem für Testumgebungen im Bereich des maschinellen Lernens und das Testen von Algorithmen neuronaler Netze. Aufgrund dessen ist sie optimal für unser Projekt geeignet.

## 3.1 Training

Voraussetzung für das maschinellen Lernen mit neuronalen Netzen ist das Anlernen mit Hilfe von Testdaten. Hierzu versuchen wir ein möglichst weit gefächertes Spektrum der handgeschriebenen Ziffern, die uns als kleine Bilder mit 28 x 28 Pixeln vorliegen, zu erzeugen, indem wir nicht nur die Ziffern, die wir zum Anlernen verwenden, zufällig auswählen, sondern auch ihre Position variieren, indem wir den Bildbereich in Höhe sowie Breite um eine beliebige Varianz erweitern und so das Netz auf weitere Variation trainieren. Dies ist anschaulich und beispielhaft in Abbildung 3.1 dargestellt, die mit Hilfe von PyPlot generiert wurde.

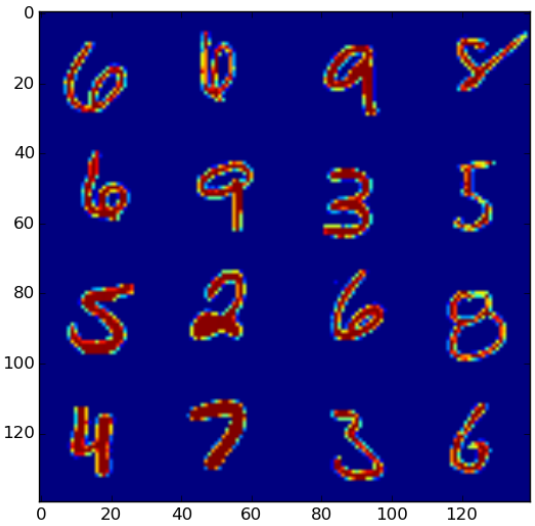

*Abb. 1.5: Beispiel für zufällig generierte Trainingsdaten*

Nunmehr ist es uns möglich, den Caffe-Server mit solchen generierten Trainingsdaten auszustatten und den Anlernvorgang durchzuführen.