Hochschule für Technik und Wirtschaft Dresden

Fakultät Informatik/Mathematik

Bachelorarbeit

Thema:

Merkmalserkennung von Gebäuden und Grundstücken in Satellitenbildern mittels Deeplearning

Vorgelegt von: Sebastian Mischke
Dorfstraße 8, 01257 Dresden
geb. am 09.11.1995 in Dresden
Bibliotheksnummer: 37612

Studiengang: Medieninformatik

Betreuender Prüfer: Prof. Dr. Marco Block-Berlitz

Zweitgutachter: Prof. Dr. Hans-Joachim Böhme

Externer Betreuer: Ann-Christin Storms
New Web Technology GmbH

Letzte Änderung: 22. Juli 2017

Abgabetermin: 27. Juli 2017

Inhaltsverzeichnis

Nomenclature

API Eine Programmierschnittstelle, genauer Schnittstelle zur Anwendungsprogrammierung, häufig nur kurz API genannt (englisch application programming interface, wörtlich ‚Anwendungsprogrammierschnittstelle'), ist ein Programmteil, der von einem Softwaresystem anderen Programmen zur Anbindung an das System zur Verfügung gestellt wird.

CNN Convolutional Neural Network
Ein Convolutional Neural Network (CNN oder ConvNet), zu Deutsch etwa „faltendes neurales Netzwerk", ist ein feedforward künstliches neuronales Netz. Es handelt sich um ein von biologischen Prozessen inspiriertes Konzept im Bereich des maschinellen Lernens (engl. machine learning). Convolutional Neural Networks finden Anwendung in zahlreichen, modernen Technologien der künstlichen Intelligenz, vornehmlich bei der maschinellen Verarbeitung von Bild- oder Audiodaten.

LSTM Long short-term memory
Long short-term memory (LSTM) bezeichnet im maschinellen Lernen einen Typ von rekurrenten neuronalen Netzen. Sie wurden 1997 von Sepp Hochreiter und Jürgen Schmidhuber in einer Veröffentlichung vorgestellt. Im Gegensatz zu traditionellen rekurrenten Netzen können LSTMs längere zeitlich verzögerte Effekte z. B. für Klassifizierungsaufgaben berücksichtigen und effektiv trainiert werden.

MLP A multilayer perceptron (MLP) is a feedforward artificial neural network model that maps sets of input data onto a set of appropriate outputs. An MLP consists of multiple layers of nodes in a directed graph, with each layer fully connected to the next one. Except for the input nodes, each node is a neuron (or processing element) with a nonlinear activation function. MLP utilizes a supervised learning technique called backpropagation for training the network. MLP is a modification of the standard linear perceptron and can distinguish data that is not linearly separable.

Abbildungsverzeichnis

Algorithmenverzeichnis

Tabellenverzeichnis

Zusammenfassung

Die stetig wachsende Menge an verfügbaren Trainingsdaten, sowie die ansteigende Rechnerleistung verbessert die Ergebnisse, die mittels Deeplearning erreicht werden können, immer mehr. Doch

In dieser Arbeit wird ein System entwickelt, das sich basierend auf einer Adressliste automatisiert alle benötigten Satellitenbilder herunterlädt, ein an die Ein- und Ausgabedaten angepasstes künstliches neuronales Netz erstellt und dieses anschließend für seine Aufgabe im Bereich der Klassifizierung beziehungsweise der Merkmalserkennung trainiert. Die Qualität der Parameter des Netzes werden dabei stetig evaluiert durch dem Netz bisher unbekannte Validierungsdaten und bei Verbesserungen für eine spätere Verwendung in Anwendungen inklusive Netzarchitektur lokal gespeichert.

Das soll die Verwendung von Deeplearning für Satellitenbilder erleichtern und damit die Möglichkeiten für zukünftige Anwendungsfälle vereinfachen.

1 Einleitung und Motivation

„Ein Bild sagt mehr als tausend Worte“. Doch gilt dies für alle Bilder? Und was sind das für Worte? Mit dem Aufkommen von Deeplearning und der hochwertige Möglichkeiten der Bildanalyse eines Convolutional Neural Networks ist die Wissenschaft der Feature-Detection in den letzten Jahren weit vorangeschritten.

2 Konkretisierung der Aufgabenstellung

Ziel dieser Arbeit ist es, ein System zu entwickeln, mit dem Klassifikationen und Merkmalserkennungen in Satellitenbildern mittels Deeplearning leicht umzusetzen sind. Dabei sollte sich das System einfach und schnell an unterschiedliche Eingabeformate der Daten anpassen lassen.

Ebenso ist ein wichtiges Kriterium des Systems die leichte Integration in zukünftige Anwendungen.

3 Verwandte Arbeiten

3.1 ImageNet Large-Scale Visual Recognition Challenge

Die ImageNet Large-Scale Visual Recognition Challenge (ILSVRC)[1] ist wohl einer der bekanntesten Wettbewerbe im Bereich der automatisierten Bildklassifizierung. Ihr Aufbau ist dabei an die Pascal Visual Object Classes Challenge[2] angelehnt. Die Trainingsdaten bestehen aus Bilddaten und der Klasseninformation zu einem Objekt, welches jeweils im Bild vorhanden ist. Dies stammt alles aus der ImageNet-Datenbank[3].

Mit SuperVision[4], einem Deep Convolutional Neural Network, gewannen Alex Krizhevsky, Ilya Sutskever und Geoffrey E. Hinton die ILSVRC-2012. Ihr Netz beinhaltet fünf Konvolutionsschichten, manche gefolgt von einer Max-pooling Schicht, und abschließend drei vollvernetzten Schichten mit der finalen 1000-Neuronen Schicht, die das Ergebnis der Klassifizierungen darstellt. Damit konnte eine Top-5-Fehlerquote, die die relative Häufigkeit angibt, in der das korrekte Ergebnis nicht in den ersten fünf Prädiktionen liegt, von 16,4% erreicht werden. Die Fehlerquote für das korrekte Erkennen der richtigen Klasse lag dabei bei 38,1%.

VGG16 und VGG19[5], die Gewinner des ILSVRC-2014 im Bereich „Klassifikation und Lokalisation“ benutzten einen ähnlichen Aufbau von Konvolutions-, Max-pooling- und vollvernetzten Schichten, nur das sich die Tiefe des Netzes deutlich vergrößert hat. Statt fünf besitzt das Netz nun 13 beziehungsweise 16 Konvolutionsschichten. Damit konnte die Top-5-Fehlerquote auf 7,2% und die Top-1-Fehlerquote auf 24,4% gesenkt werden.

3.2 Multi-column Deep Neural Networks

Eine Alternative zum linearen neuronalen Netzwerk sind die mehrspaltigen neuronalen Netzwerke[6]. Die Architektur besteht dabei aus mehreren herkömmlichen neuronalen Netzen, welche parallel zueinander angeordnet sind. Dabei ist anzumerken, dass diese mehreren Netze zunächst einzeln trainiert werden und erst für die spätere Anwendung zusammengeschlossen werden.

Auch dann noch verarbeiten die Teilnetze die Eingabedaten unabhängig voneinander. Für das abschließende Ergebnis des mehrspaltigen Netzes bildet man den Durchschnittswert der Einzelergebnisse der Teilnetze. Dies erbrachte in den durchgeführten Experimenten eine relative Verbesserung der Klassifizierungsgenauigkeit zwischen 26% und 72% im Vergleich zu den bisher besten Ergebnissen.

3.3 PlaNet

Im Projekt „PlaNet - Photo Geolocation with Convolutional Neural Networks“[7] wurde versucht, den Aufnahmeort eines Fotos lediglich anhand des Fotos zu erkennen. Dafür wurde die Weltkugel in mehrere Zellen eingeteilt und die Bilder jeweils der Zelle zugeordnet, in der ihr tatsächlicher Aufnahmeort liegt. Damit konnte in etwa 30% das korrekte Land erkannt werden und die Genauigkeitsrate für Städte lag bei rund 10%. Das reichte, um bisherige Modelle, welche noch auf handeingetragenen Merkmalen basierten, mit einen signifikanten Abstand zu überbieten. Außerdem war man in der Lage in einem Experiment beim Onlinespiel GeoGuessr gegen erfahrene Menschen zu gewinnen.

Das System wurde anschließend noch verbessert, indem das bestehende Convolutional Neural Network mit einem Long short-term memory (LSTM) kombiniert wurde, um anstatt eines einzelnen Bildes ein ganzes Bilderalbum zu klassifizieren.

3.4 isprs 2D Semantic Labeling Contest

Die Internationale Gesellschaft für Photogrammetrie und Fernerkundung trägt einen 2D Semantic Labeling Wettbewerb[8] für Luftaufnahmen aus. Darin müssen anhand von Orthofotos, das heißt verzerrungsfreien und maßstabsgetreuen Abbildung der Erdoberfläche, welche durch Luft- und Satellitenaufnahmen erstellt wurden, sowie digitalen Höhenmodellen bestimmte Objekte und Kategorien erkannt werden. Bei diesen Kategorien handelt es sich um Flächenversiegelungen, also Straßen, Parkplätze und ähnliches, Gebäude, niedrige Vegetation, Bäume, Fahrzeuge und die gesonderte Klasse für Sonstiges, wozu Wasser oder andere Flächen gehören, die sich den bisherigen Gruppen nicht zuordnen ließen. Da es sich hier um die Klassifizierung größerer Bereiche handelt, werden die Klasseninformationen farbkodiert in Bildern gespeichert, die der identischen lokalen Zuordnung

entsprechen, wie die dazugehörigen Orthofotos.

4 Verwendete Technik

Die Möglichkeiten der Technik sind vielzählig. Von der Programmiersprache, über die verwendete Netzstruktur, die Bibliotheken und Module, bis hin zu Interfacespezifika und Visualisierungen.

4.1 Python

Die verwendete Programmiersprache ist Python.

Python besitzt viele Vorteile gegenüber anderen Programmiersprachen. Es ist leicht zu lesen, wodurch man die Logik und den Ablauf eines Programmes schneller erkennt. Es ist unabhängig vom Betriebssystem und die große Anzahl an open-source Bibliotheken ermöglicht eine leichte Umsetzung zu nahezu jeder Problematik. Außerdem besitzt Python die größte Anzahl an Deeplearning-Bibliotheken, und wird auch von großen Unternehmen und Einrichtungen in diesem Bereich unterstützt und vorangetrieben, wie TensorFlow[9] von Google, CNTK[10] von Microsoft oder cuDNN[11] von NVIDIA.

4.2 Convolutional Neural Networks

Wie SuperVision[4] und VGG16[5] zeigen konnten, ist die Architektur eines Convolutional Neural Networks mit abschließenden vollvernetzten Schichten am besten geeignet, um Bilddaten auszuwerten. Und selbst die mehrspaltigen neuronalen Netze[6] haben diese Grundstruktur intern beibehalten.

4.3 Keras

Als Deeplearning Framework wurde Keras[12] verwendet. Es benotigt als Grundlage entweder TensorFlow[9], CNTK[10] oder Theano[13]. Dadurch ist die Installation aufwändiger als bei Bibliotheken, die keine zusätzlichen Voraussetzungen besitzen, jedoch erlaubt es aber auch die Implementierung und damit die Vorteile von allen dreien zu benutzen.

4.4 matplotlib

Nahezu alle Visualisierungen wurden mit matplotlib, einer Python Bibliothek für graphische Darstellungen, gemacht. Damit lassen sich nicht nur Grafiken erzeugen, direkt anzeigen oder abspeichern, sondern auch in anderen GUI-Bibliotheken

wie `GTK+`, `Qt`, `wxWidgets` oder in unserem Fall Tkinter verwenden.

4.5 Tkinter

Da Python keine eigene GUI besitzt, muss zum Programmieren eines Interfaces eine zusätzliche GUI-Bibliothek wie Tkinter[14] verwendet werden.

5 Gesamtplan

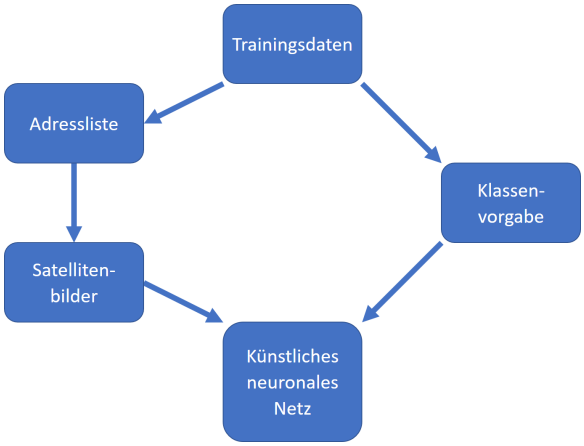

Abbildung 1: Datenflussdiagramm

Das System zur Merkmalserkennung besteht aus drei Kernbausteinen. (siehe Abbildung 1)

- Den Satellitenbildern, die als Informationsgrundlage und Input-Daten für den Klassifizierer genutzt werden
- Dem künstlichen neuronalen Netz, welches die Input-Daten analysiert und klassifiziert
- Der Trainingsvorgabe, welche zum Trainieren des Netzes benutzt wird.

Algorithmus 1 Hauptprozedur des Systems

```
1  # Load data
2  images, teach = load_csv("data.csv")
3
4  # Create net
5  model = create_net(images, teach)
6
7  # Create checkpointer
8  cp = ModelCheckpoint(
9      filepath='model.hdf5'
10 )
11
12 # Train the net
13 model.fit(images, teach,
14     batch_size=batchsize,
15     epochs=epochs,
16     validation_split=split,
17     callbacks=[cp]
18 )
```

5.1 Trainingsdaten

Die Trainingsdaten geben dem künstlichen neuronalen Netz vor, welche Aufgabe es zu erfüllen hat. Dabei bestehen die Trainingsdaten aus zwei Teilen.

Den Satellitenbildern, die die Eingabedaten des Netzes sind und welche auch für spätere Anwendungsfälle benötigt werden. Deren genauere Erzeugung wird in Abschnitt 5.2 erklärt.

Der zweite Teil sind die Klasseninformationen der jeweiligen Bilder, welche die gewünschten Ausgabedaten des Netzes sind. Das Erzeugen der Klasseninformationen ist in der Regel ein aufwändiger Prozess und stark vom gewünschten zu erkennenden Merkmal abhängig. Es werden größere Mengen an Trainingsdaten benötigt, um ein Netz zu trainieren. Und wenn es für die gewünschte Aufgabe keine bereits existierende Datenquelle gibt, der man diese Menge an Informationen entnehmen kann, dann müssen sie notfalls eben händisch erzeugt werden. Dafür beinhaltet das System ein Interface, welches in Abbildung 2 zu sehen ist, undzeilenweise die Adressen einer CSV-Datei benutzt, um aus ihnen Satellitenbildern ebenfalls basierend auf Abschnitt 5.2 zu erzeugen, und diese anzeigt. Der Benutzer kann anschließend aus einer Auswahl an Klasseninformationen die dem Bild zugehörige auswählen. Seine Entscheidung wird danach zusammen mit der momentanen Adresse in einer neuen CSV-Datei abgespeichert.

5.2 Satellitenbilder

Die Satellitenbilder bilden die Grundlage der Eingabedaten für das Netz. Da eine eigenständige Erzeugung einen viel zu hohen Aufwand bedeuten würde, werden stattdessen Bilder verwendet, die mittels der Google Static Maps API erzeugt wurden. Dafür wird eine URL mittels Algorithmus 2 erzeugt, die die Einstellungen des zu erzeugenden Bildes beinhaltet. Zum einen der Mittelpunkt in Form einer Adresse oder einer GPS-Koordinate. Dies ist die Kerninformation und das sich später einzige ändernde Merkmal innerhalb einer Bilderreihe für einen Trainingsprozess. Dann die Größe des Bildes angegeben in Pixeln. Hierbei liegt das Maximum bei 640x640, deswegen wird dies auch die Standardeinstellung. Anschließend die Zoom-Stufe, die Größe eines Objektes innerhalb eines Bildes angibt. Die offizielle Google Maps API Dokumentation[1] gibt eine Tabelle für die Detailebenen an, die man basierend auf diesem Parameter erwarten kann. Außerdem gibt es den Parameter des Kartentypen. Da es lediglich um die Informationsgewinnung aus Satellitenaufnahmen geht, ist hier immer satellite ausgewählt.

Abbildung 2: Anwendung zum händischen Erzeugen von Trainingsdaten

Algorithmus 2 Erzeugen der URL für die Google Maps API

```
1  url = "http://maps.google.com/maps/
       api/staticmap"
2  if centermode == "xy":
3      # Latitude and Longitude
4      url += "?center=" + y + "," + x
5  elif centermode == "address":
6      # Address
7      url += "?center=" + address
8  url += "&size=" + w + "x" + h
9  url += "&zoom=" + zoom
10 url += "&maptype=" + maptype
```

Die mittels dieser URL erzeugten Bilder werden lokal gespeichert und basierend auf ihren Parametern benannt. Dadurch könnten sie vor dem

[1] https://developers.google.com/maps/documentation/javascript/tutorial#vergrerungsstufen

Trainingsprozess noch zugeschnitten oder auf eine andere Größe skaliert werden. Außerdem verhindert es ein mehrfaches Herunterladen des gleichen Bildes.

5.3 Künstliche Datenmengenvergrößerung

Sollte die Menge an Trainingsdaten für die gewünschte Aufgabe des Netzes zu klein sein, ist das Risiko eines Overfittings und der damit einhergehenden schlechten Genauigkeitsrate für die Validierungsdaten sehr hoch.

Eine sehr einfache Variante um die Menge der Trainingsdaten künstlich zu erhöhen, ist das geringe Verändern der Satellitenbilder an sich bei gleichbleibender Klasseninformation. Die genaue Abwandlung eines Bildes ist dabei zufällig, aber durch gewisse Rahmenbedingungen eingeschränkt. Wie in Algorithmus 3 gezeigt, ist dabei eine Vielzahl von Operationen, wie das Spiegeln, Drehen, Verschieben, Strecken oder Vergrößern des Bildes möglich.

Algorithmus 3 ImageDataGenerator

```
from keras.preprocessing.image
    import ImageDataGenerator

# Verändern der Bilddaten
datagen = ImageDataGenerator(
    rotation_range=10,
    width_shift_range=0.1,
    height_shift_range=0.1,
    rescale=1. / 255,
    shear_range=0.1,
    zoom_range=0.1,
    horizontal_flip=True,
    vertical_flip=True,
    fill_mode='reflect'
)
```

Die Bilddaten werden hierbei aber nicht mehr über eine externen Datei geladen, die sämtliche Adressen der Trainingsdaten enthält, sondern direkt in Form von Bilddateien aus einem Ordner, wie in Algorithmus 4 zu erkennen.

Für das Erzeugen der Bilddateien ist dafür aber immer noch eine seperate Adressdatei notwendig und erfolgt wie in Abschnitt 5.2 beschrieben. Die Klasseninformation werden nun in Form von Unterordnern übermittelt. Jeder Ordner steht dabei für eine eigenständige Klasse. Deshalb ist diese

Algorithmus 4 flow_from_directory

```
# Erzeugen der Trainingsdaten
generator = datagen.
    flow_from_directory(
    'data/train_data',
    target_size=(width, height),
    batch_size=batchsize,
    class_mode=classmode,
    shuffle=True
)
```

Methode auch nur für Versuche mit klaren Klassenzuordnungen anwendbar.

Diese Variante der Datenmengenvergrößerung wurde auch in „Building powerful image classification models using very little data“[15] verwendet.

5.4 Erstellen des künstlichen neuronalen Netzes

Die Grundstruktur des Netzes basiert auf der Vorgaben von VGG16[5]

Das neuronale Netz besteht es drei Teilen. Einem CNN, welches die Satellitenbilder als Input-Daten bekommt und diese auswertet, einem MLP, welches zusätzliche Metadaten des Gebäudes bzw. Grundstückes als Input-Daten bekommt, und einem MLP, welches die Output-Daten des CNN und des MLP als Input-Daten bekommt.

Best Practices for Convolutional Neural Networks Applied to Visual Document Analysis[16]

Dropout: a simple way to prevent neural networks from overfitting[17]

5.5 Training des Netzes

Da das Netz keine Farbbilder als Eingabe akzeptieren kann, müssen die Bilder vorher in drei 2D-Arrays für den Rot-, Grün- und Blaukanal umgewandelt werden.

Nachdem nun die Trainingsdaten geladen und das neuronale Netz erstellt wurde, müssen die Gewichte des Netzes an das gewünschte Ziel angepasst werden. Dies geschieht, indem ein Optimierer, in unserem Fall ein Stochastischer Gradientenabstieg[18], in mehreren Epochen die Klassifizierung des Netzes an die Trainingsvorgabe anpasst.

5.6 Bewertung des Netzes

Nach jeder Trainingsepoche werden die Validierungsdaten vom Netz klassifiziert und die Ergebnisse mit der Trainingsvorlage verglichen. Der sich daraus ergebende Genauigkeitswert wird dann mit dem bisherigen Bestwert verglichen. Sollte die neue Klassifizierung besser sein als der bisherige Spitzenwert, so werden die Gewichte des Netzes gespeichert und der neue Bestwert wird sich gemerkt. Dadurch soll ein Overfitting des Netzes vermieden werden.

Die Methode, nach der der Genauigkeitswert berechnet wird, hängt dabei vom Format der Ausgabedaten, und damit auch von der Aktivierungsart der letzten Schicht des Netzes ab. Wenn es sich um eine

5.7 Visualisierung von Klassifizierungen

Für die Darstellung der Ergebnisse der Klassifizierung des Netzes gibt es verschiedene Möglichkeiten.

Eine Variante ist die Darstellung basierend auf GPS-Koordinaten. Dabei wird mit matplotlib, siehe Abschnitt 4.4, ein Streudiagramm erstellt. Die x- und y-Werte bestehen dabei aus den Längen- und Breitengraden der GPS-Koordinaten. Die Ergebnisse der Klassifizierung werden dann in der Farbe der Punkte kodiert. Ein Beispiel dafür ist in Abbildung 6 dargestellt.

6 Experimente

Um das System zu testen, wurden mehrere Experimente mit verschiedenen Datengrundlagen für die Eingabedaten, sowie verschiedenen Formaten beziehungsweise Typen von Ausgabedaten durchgeführt.

6.1 GPS-Koordinaten

Der erste Versuch besteht daraus, die GPS-Koordinaten, das heißt sowohl Längen-, als auch Breitengrad, eines Satellitenbildes zu bestimmen. Dafür sollen keine zusätzliche Metadaten über das Bild bekannt sein. Die verfügbaren Daten zum Trainieren des Netzes basieren dabei auf den 6337 Einträgen der Datenbank "Geografische Längen- und Breitengrade deutscher Städte und Gemeinden"[2], deren Verteilung in Abbildung 3 dargestellt ist.

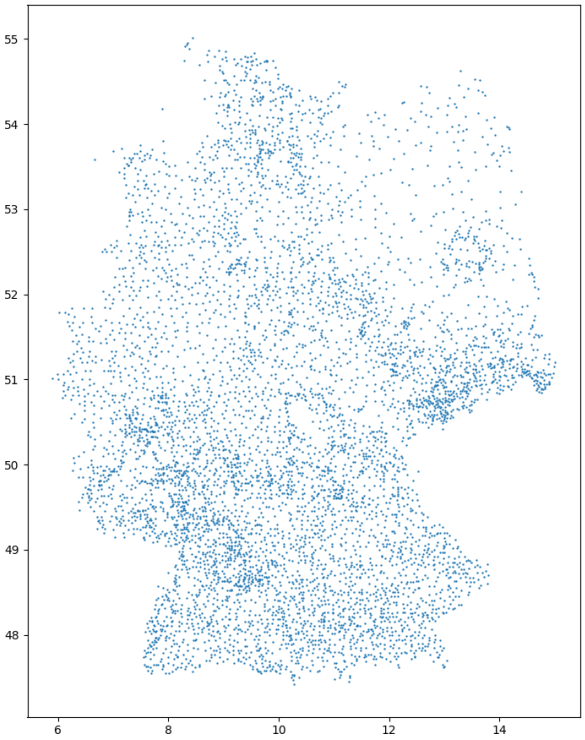

Abbildung 3: Deutsche Städte und Gemeinden

Bei einer Trainings-Validierungs-Aufteilung von 10% ergibt dies eine Anzahl von 5703 Trainingsdaten und 634 Validierungsdaten.

[2] http://www.fwiegleb.de/geodat.htm

Die mittleren absoluten Fehler der Trainings- und Validierungsdaten lagen bei 2,602 und 1,534. Der Verlauf dieses Fehlers während des Trainings des Netzes ist in Abbildung 4 dargestellt.

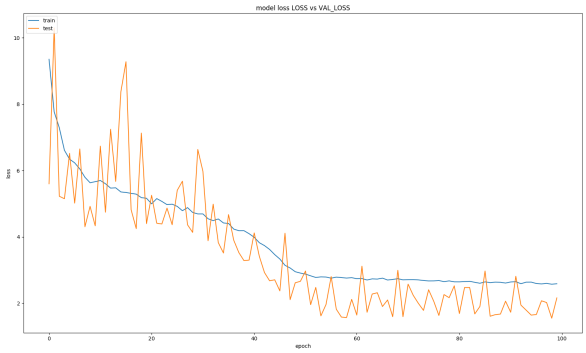

Abbildung 4: Mittlerer absoluter Fehler während der Trainingsepochen

Die genauere Verteilung der Ergebnisse ist Abbildung 5 und Tabelle 1 zu entnehmen.

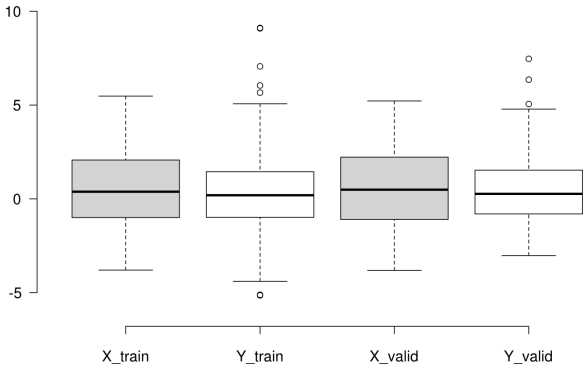

Abbildung 5: Abweichungen der Netzvorhersagen vom Vorgabewert

6.2 Postleitzahl

Im Gegensatz zu Längen- und Breitengrad, sind Postleitzahlen in Deutschland nur bedingt geordnet. Es gilt zwar das Prinzip, dass zwei zahlenmäßig nah beieinander liegende Postleitzahlen sich häufig auch örtlich nah sind, jedoch ist diese Zuordnung nicht konsistent. Deswegen sollte es für ein Netz auch entsprechend schwerer sein, nur anhand eines Satellitenbildes eines Hauses die Postleitzahlen von eben diesem zu bestimmen.

Die Trainingsdaten bestehen aus 64390 Häusern in Dresden, deren räumliche Anordnung sowie Zuordnung zu einer Postleitzahl in Abbildung 6 dargestellt ist.

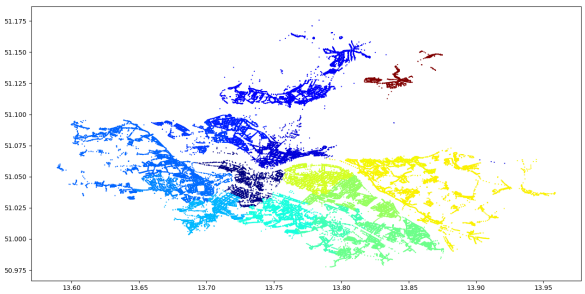

Abbildung 6: Postleitzahlenbereiche in Dresden

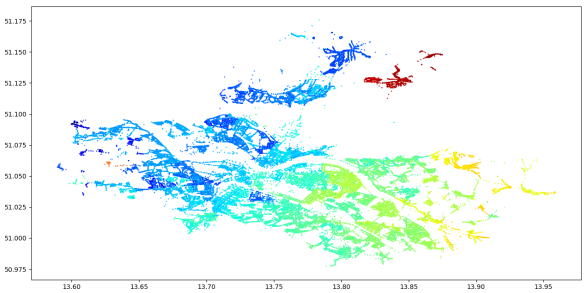

Abbildung 7: Ergebnisse der Klassifizierung des Netzes

6.3 Schulen - Wohnhäuser

Das dritte Experiment wird die Klassifizierung von Häusern beinhalten. Dafür soll ein Netz trainiert werden, welches zwischen Wohnhaus und Schule unterscheiden kann. Als Datengrundlage wird hierfür eine Liste mit allen 204 Schulen in Dresden[3], sowie 204 zufällig ausgewählte Wohnhäusern, ebenfalls aus dem Raum Dresden.

6.4 Schularten

Das letzte Experiment beinhaltet die Klassifizierung von Schulen in Sachsen nach den drei Klassen Grundschule, Oberschule und Gymnasium. Die dafür verwendeten Trainingsdaten basieren auf der offiziellen Sächsischen Schuldatenbank[4], herausgegeben vom Sächsischen Staatsministerium für Kultus.

[3] https://www.dresden.de/de/leben/schulen/schulen-in-dresden.php

[4] https://schuldatenbank.sachsen.de/index.php?id=2

7 Ergebnisse

Das System ließ sich einfach, schnell und problemlos an die verschiedenen Anforderungen der Experimente anpassen.

8 Ausblick

Die wohl größte Erweiterung für das erzeugte System wäre die Umsetzung einer Anwendung, die ein trainiertes Netz automatisch als Grundlage für die Klassifizierung von Satellitenbildern nimmt. Die Satellitenbilder könnten ebenfalls auf Bedarf mittels der Google Maps API erzeugt werden und auf vom Nutzer eingegebenen Adressen basieren.

Literatur

[1] O. Russakovsky, J. Deng, H. Su, J. Krause, S. Satheesh, S. Ma, Z. Huang, A. Karpathy, A. Khosla, M. Bernstein, A. C. Berg, and L. Fei-Fei, "Imagenet large scale visual recognition challenge," *International Journal of Computer Vision*, vol. 115, no. 3, pp. 211–252, Dec 2015. [Online]. Available: https://doi.org/10.1007/s11263-015-0816-y

[2] M. Everingham, L. Van Gool, C. K. Williams, J. Winn, and A. Zisserman, "The pascal visual object classes (voc) challenge," *International journal of computer vision*, vol. 88, no. 2, pp. 303–338, 2010.

[3] J. Deng, W. Dong, R. Socher, L.-J. Li, K. Li, and L. Fei-Fei, "Imagenet: A large-scale hierarchical image database," in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 248–255.

[4] A. Krizhevsky, I. Sutskever, and G. E. Hinton, "Imagenet classification with deep convolutional neural networks," in *Advances in Neural Information Processing Systems 25*, F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, Eds. Curran Associates, Inc., 2012, pp. 1097–1105. [Online]. Available: http://papers.nips.cc/paper/4824-imagenet-classification-with-deep-convolutional-neural-networks.pdf

[5] K. Simonyan and A. Zisserman, "Very deep convolutional networks for large-scale image recognition," *arXiv preprint arXiv:1409.1556*, 2014.

[6] D. Ciregan, U. Meier, and J. Schmidhuber, "Multi-column deep neural networks for image classification," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3642–3649.

[7] T. Weyand, I. Kostrikov, and J. Philbin, "Planet-photo geolocation with convolutional neural networks," in *European Conference on Computer Vision*. Springer, 2016, pp. 37–55.

[8] I. G. für Photogrammetrie und Fernerkundung. 2d semantic labeling contest. [Online]. Available: http://www2.isprs.org/commissions/comm3/wg4/semantic-labeling.html

[9] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard *et al.*, "Tensorflow: A system for large-scale machine learning." in *OSDI*, vol. 16, 2016, pp. 265–283.

[10] F. Seide and A. Agarwal, "Cntk: Microsoft's open-source deep-learning toolkit," in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*. ACM, 2016, pp. 2135–2135.

[11] S. Chetlur, C. Woolley, P. Vandermersch, J. Cohen, J. Tran, B. Catanzaro, and E. Shelhamer, "cudnn: Efficient primitives for deep learning," *arXiv preprint arXiv:1410.0759*, 2014.

[12] F. Chollet, "Keras," https://github. com/fchollet/keras, 2015.

[13] J. Bergstra, O. Breuleux, P. Lamblin, R. Pascanu, O. Delalleau, G. Desjardins, I. Goodfellow, A. Bergeron, Y. Bengio, and P. Kaelbling, "Theano: Deep learning on gpus with python," 2011.

[14] J. E. Grayson, *Python and Tkinter programming*. Manning,, 2000.

[15] F. Chollet, "Building powerful image classification models using very little data," *Retrieved December*, vol. 13, p. 2016, 2016.

[16] P. Y. Simard, D. Steinkraus, J. C. Platt *et al.*, "Best practices for convolutional neural networks applied to visual document analysis." in *ICDAR*, vol. 3, 2003, pp. 958–962.

[17] N. Srivastava, G. E. Hinton, A. Krizhevsky, I. Sutskever, and R. Salakhutdinov, "Dropout: a simple way to prevent neural networks from overfitting." *Journal of Machine Learning Research*, vol. 15, no. 1, pp. 1929–1958, 2014.

[18] L. Bottou, "Large-scale machine learning with stochastic gradient descent," in *Proceedings of COMPSTAT'2010*. Springer, 2010, pp. 177–186.

	X_train	Y_train	X_valid	Y_valid
Maximum	5.48	5.07	5.22	4.78
Oberes Quartil	2.07	1.45	2.22	1.53
Median	0.38	0.19	0.49	0.27
Unteres Quartil	-1.00	-0.99	-1.10	-0.80
Minimum	-3.80	-4.40	-3.81	-3.02
Anzahl Datenpunkte	5703	5703	634	634

Tabelle 1: Ergebnisse des x-y-Koordinatenversuchs

Algorithmus 5 Funktion zum Erstellen des künstlichen neuronalen Netzes

```
1  def create_net(X_train, Y_train):
2      from keras.models import Sequential
3      from keras.layers import (Flatten, Dense, Convolution2D, MaxPooling2D,
           ZeroPadding2D, Dropout)
4
5      model = Sequential()
6      model.add(ZeroPadding2D((1, 1), input_shape=X_train[0].shape[1:]))
7      model.add(Convolution2D(16, (3, 3), activation='relu'))
8      model.add(ZeroPadding2D((1, 1)))
9      model.add(Convolution2D(16, (3, 3), activation='relu'))
10     model.add(MaxPooling2D((2, 2), strides=(2, 2)))
11
12     model.add(ZeroPadding2D((1, 1)))
13     model.add(Convolution2D(32, (3, 3), activation='relu'))
14     model.add(ZeroPadding2D((1, 1)))
15     model.add(Convolution2D(32, (3, 3), activation='relu'))
16     model.add(MaxPooling2D((2, 2), strides=(2, 2)))
17
18     model.add(ZeroPadding2D((1, 1)))
19     model.add(Convolution2D(64, (3, 3), activation='relu'))
20     model.add(ZeroPadding2D((1, 1)))
21     model.add(Convolution2D(64, (3, 3), activation='relu'))
22     model.add(ZeroPadding2D((1, 1)))
23     model.add(Convolution2D(64, (3, 3), activation='relu'))
24     model.add(MaxPooling2D((2, 2), strides=(2, 2)))
25
26     model.add(ZeroPadding2D((1, 1)))
27     model.add(Convolution2D(128, (3, 3), activation='relu'))
28     model.add(ZeroPadding2D((1, 1)))
29     model.add(Convolution2D(128, (3, 3), activation='relu'))
30     model.add(ZeroPadding2D((1, 1)))
31     model.add(Convolution2D(128, (3, 3), activation='relu'))
32     model.add(MaxPooling2D((2, 2), strides=(2, 2)))
33
34     model.add(Flatten())
35     model.add(Dense(1024, activation='relu'))
36     model.add(Dropout(0.1))
37     model.add(Dense(1024, activation='relu'))
38     model.add(Dropout(0.1))
39     model.add(Dense(Y_train.shape[1], activation='linear'))
40
41     from keras.optimizers import SGD
42     SGD = SGD(lr=0.0001, momentum=0.0, decay=0.0, nesterov=False)
43     model.compile(loss='mean_absolute_error', optimizer=SGD, metrics=['mse'])
44
45     return model
```