Julius-Maximilians-Universität Würzburg
Institut für Informatik
Lehrstuhl für Informatik I
Effiziente Algorithmen und wissensbasierte Systeme

Bachelorarbeit

Flusserkennung auf Luftaufnahmen durch maschinelles Lernen

Jonas Jung

Eingereicht am 28. August 2017

Betreuer:
Prof. Dr. Alexander Wolff
Prof. Dr. Peter Gehler

Zusammenfassung

In dieser Arbeit wird versucht ein neuronales Netz auf die Erkennung von Flüssen in Luftaufnahmen zu trainieren. Dabei wird eine Methode zur halb-automatischen Generierung von Trainingsdaten vorgestellt und das dadurch entstandene Modell auf Präzision getestet. Anschließend wird noch die Frage behandelt, wie sinnvoll die Nutzung eines Neuronalen Netzes zum Lösen dieser Aufgabenstellung ist.

Inhaltsverzeichnis

1 Einleitung

Die sogenannte künstliche Intelligenz ist auf dem Vormarsch und mit ihr die künstlichen neuronalen Netze (KNN), welche einen Ansatz bieten um Computern vorher nahezu unmögliche Aufgabenstellungen bewältigen zu lassen. Eine dieser Aufgaben ist das Erkennen von Objekten auf Fotos. Eine Aufgabe, die für Menschen trivial klingt, an der jedoch Computer seit langer Zeit verzweifelt sind und dessen Bewältigung die Tür zu vielen neuen Anwendungen öffnet. Ein Beispiel sind autonome Fahrzeuge, welche Bilder über Kameras bekommen und in Echtzeit entscheiden müssen, welche Pixel eine Straße, welche Fahrzeuge und welche andere Objekte wie Straßenschilder oder Personen abbilden.

Diese Arbeit befasst sich mit eben einer dieser Aufgaben: Die Erkennung von Flüssen auf Luftaufnahmen. Dabei soll für ein gegebenes Foto automatisch entschieden werden, welche Pixel zu einem Fluss gehören und welche nicht. Außerdem soll der sonst aufwändige Schritt der Trainingsdaten-Erstellung halb-automatisiert werden.

Ein potentielles Anwendungsbeispiel ist in der Bachelorarbeit von Felix Klesen behandelt worden, welche erörtert, wie man anhand vom Flussverlauf den Fluss identifizieren kann. Kombiniert man das mit einem Programm, welches auf einem Foto eines Flusses den Fluss isoliert, kann man ermitteln lassen um welchen Fluss es sich auf dem Foto handelt und ohne andere Lokalisierungsmethoden wie zum Beispiel GPS, die eigene Position bestimmen.

2 Stand der Technik

2.1 Was sind künstliche neuronale Netze?

Künstliche neuronale Netze sind ein Teilgebiet der künstlichen Intelligenz und versuchen komplexe Probleme durch Imitation von natürlichen neuronalen Netzen, wie sie in Gehirnen von Menschen vorkommen, zu lösen. In der Regel handelt es sich dabei um Probleme, die für Menschen trivial wirken, bei der die herkömmliche Informatik allerdings an ihre Grenzen stößt [1], wie zum Beispiel im Falle dieser Arbeit das Erkennen und Zuordnen von Mustern auf Bildern, konkreter noch das Erkennen von Flüssen auf Luftaufnahmen. Das Erkennen von Mustern und Zuordnen von Pixeln wird als *semantic segmentation* bezeichnet.

Für Menschen ist es etwas Leichtes auf Luftaufnahmen Flüsse zu erkennen, da sie in ihrem Leben unzählige Beispiele und Variationen von Flüssen gesehen haben und dadurch in ihrem Kopf das Konzept "Fluss" abstrahiert haben. Mit diesem Konzept werden unzählige von Eigenschaften verbunden, wie zum Beispiel, dass es sich um einen zusammenhängenden Wasserkörper handeln muss, der eine gewisse Breite, Tiefe und Länge besitzen, in eine Richtung fließen und natürlich entstanden sein muss. Nur wenn alle diese Eigenschaften auf ein Objekt zutreffen, erkennt ein Mensch dieses als einen Fluss, ansonsten klassifiziert er es als einen Bach, einen See, einen Kanal oder etwas anderes. Das gleiche gilt für Fotos von Flüssen: Nur wenn eine Vielzahl von Eigenschaften zutrifft, erkennen wir eine Pixelgruppe als einen Fluss.

Damit ein Computerprogramm diese Klassifizierung vornehmen kann, muss ihm dieses Konzept eines Flusses beigebracht werden. In der herkömmlichen Informatik bedeutet das, dass auf jede einzelne Eigenschaft einzeln überprüft werden muss. Das Aufschreiben jeder einzelnen Eigenschaft ist allerdings nicht praktikabel, da es zum einen zu viele davon gibt und zum anderen diese Eigenschaften im Unterbewusstsein abgespeichert sind. Wir überprüfen automatisch auf diese Eigenschaften, können aber die "Liste" der Eigenschaften nicht ohne weiteres aus unserem Kopf extrahieren.

(a) Eingabe **(b)** Gewünschte Ausgabe

Abbildung 2.1: Trainingspaar für ein Neuronales Netz.

Im Gegensatz zu einem Programm der herkömmlichen Informatik ist ein KNN mit Hilfe von Trainingsdaten in der Lage sich das Konzept eines Flusses selbst beizubringen. Dafür werden Eingangs- und Ausgangsdaten als Paare in das neuronale Netz gegeben. Ein Beispiel für ein Trainingspaar wird in Abbildung 2.1 gezeigt. Im Falle der Flusserkennung ist eine Eingangsdatei eine Luftaufnahme und die entsprechend gewünschte Ausgangsdatei ist die Summe aller Pixel der Luftaufnahme, die einen Fluss darstellen. Durch eine ausreichend vielfältige und umfangreiche Eingabe von Trainingspaaren kann ein entsprechendes KNN die Muster und Eigenschaften, die einen Fluss ausmachen, isolieren und speichern. Ist das Training abgeschlossen, überprüft das fertige Modell die neuen Eingangsdateien auf die Eigenschaften eines Flusses, die es durch das vorangegangene Training gelernt hat und gibt im besten Fall die Pixel wieder, die zu einem Fluss gehören.

KNNs sind eine Verkettung aus künstlichen Neuronen, die einen Wert annehmen, mit Hilfe einer Aktivierungsfunktion einen Wert ausgeben und diesen Wert über Gewichtungen an andere künstliche Neuronen weiterleiten [2]. Sei x_i der Eingangswert des Neurons i, y_i der Ausgangswert des Neurons i, $f_i(x)$ die Aktivierungsfunktion des Neurons i und ω_{ij} das Gewicht der Verbindung von Neuron i zu Neuron j. Dann gilt für ein beliebiges Neuron j:

$$y_j = f_j(x_j)$$

und

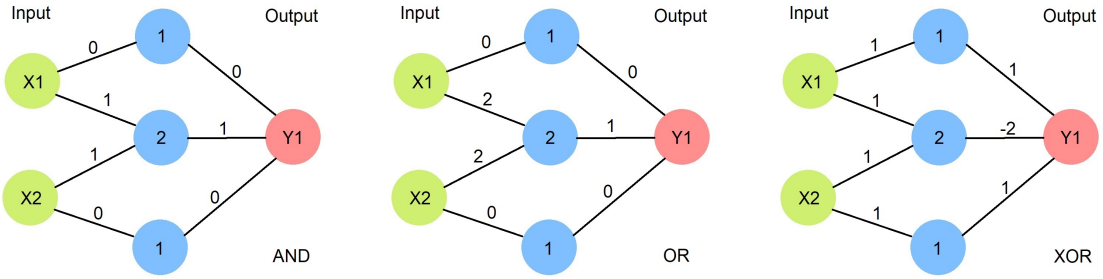
$$x_j = \sum_i f_i(x_i) \cdot \omega_{ij}$$

woraus folgt

$$y_j = f_j\left(\sum_i y_i \cdot \omega_{ij}\right) \tag{2.1}$$

für alle Neuronen i, dessen Ausgang in den Eingang des Neurons j führen.

Je nach Anzahl und Vernetzung der Neuronen können unterschiedliche Zusammenhänge modelliert werden, die einen Eingangs- und einen Ausgangsvektor haben.



(a) AND-Gate **(b)** OR-Gate **(c)** XOR-Gate

Abbildung 2.2: X1 und X2 sind die beiden Inputs, Y1 der Output, die Zahlen auf den Linien die Gewichtungen und die Zahlen in den blauen Neuronen der Inputwert der erreicht werden muss, damit dieses Neuron als Output 1 ausgibt.

Abbildung 2.2 zeigt drei Beispiele, wie mit der gleichen Konfigurationen aus Neuronen und nur mit unterschiedlichen Gewichtungen, verschiedene Logik-Gatter modelliert werden können. Der Schwellwert θ, der überschritten werden muss, damit ein Neuron den Output 1 hat, kann durch die simple Aktivierungsfunktion

$$f_i(x) = \begin{cases} 1 & \text{if } x \geq \theta_i \\ 0 & \text{else} \end{cases}$$

modelliert werden. Da alle Aktivierungsfunktionen, aus in Kapitel 3.4 behandelten Gründen, differenzierbar sein müssen, kann die oben genannte Funktion durch eine differenzierbare Approximation

$$g_i(x) = \frac{x - \theta_i}{2(|x - \theta_i| + \delta)} + \frac{1}{2}$$

für ein beliebige kleines δ ersetzt werden.

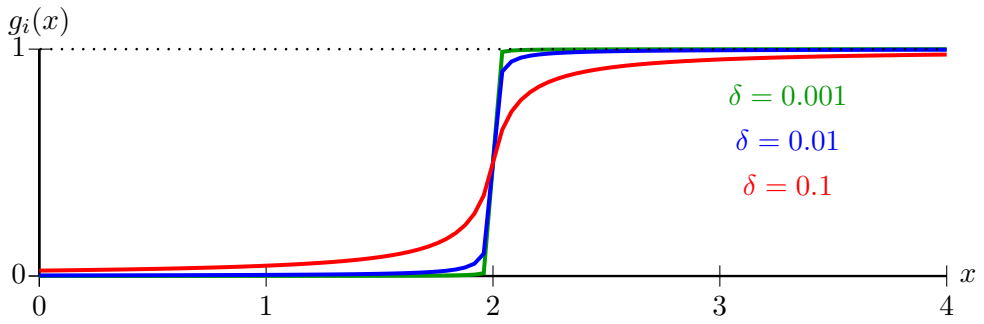


Abbildung 2.3: $g_i(x)$ für $\theta_i = 2$ und verschiedene δ .

Durch Vernetzung vieler tausend Neuronen in mehreren Ebenen können somit sehr komplexe Problemstellungen dargestellt werden.

Beim sogenannten überwachten Training wird ein Input $\vec{x} = x_1, x_2 \ldots$ für alle Eingangsneuronen zusammen mit dem gewünschten Output $\vec{y} = y_1, y_2 \ldots$ für alle Ausgangsneuronen an das KNN übergeben, welches dann die Differenz zwischen dem gewünschten und dem tatsächlichen Output $\vec{y'}$ über die Fehlerfunktion

$$E = \frac{1}{2}\|\vec{y} - \vec{y'}\|^2 \tag{2.2}$$

berechnet. Daraufhin werden die Gewichtungen ω_{ij} angepasst, sodass die Differenz verringert wird. Wie groß die Anpassung ist, hängt von der *learning rate* ab. Es gibt verschiedene Methoden zur Berechnung neuer Gewichtungen, wobei alle gemein haben, dass sie versuchen den Fehler auf das globale Minimum zu reduzieren. Welche Methode in dieser Arbeit benutzt wurde und welche Rolle genau die *learning rate* spielt wird in Kapitel 3.4 behandelt.

2.2 Andere Arbeiten zur automatischen Flusserkennung

Das Benutzen von neuronalen Netzen zur automatischen Flusserkennung ist keine neue Idee und wurde bereits erfolgreich getestet. Casado und Gonzalez[3] gelang es mit Hilfe von hochauflösenden Drohnenbildern eines Flusses, ein KNN auf die Erkennung der verschiedenen Komponenten eines Flusses wie Sandbänke, Sträucher am Ufer oder das Wasser selbst zu trainieren. Es wurde eine Genauigkeit von 81% erzielt, wobei 92,6% der Pixel im tiefen Gewässer und 58,8% der Pixel im seichten Gewässer als solche klassifiziert wurden. Die beiden großen Unterschiede zu dieser Arbeit bestehen darin, dass Casado und Gonzalez sehr hochauflösende Fotos aus geringer Höhe benutzen konnten, dafür allerdings mehr als nur zwei Klassen klassifiziert haben.

Ähnlich ist die Arbeit von Goswami, Gakhar und Kaur[4], in der es um das Identifizieren von Wasserkörpern auf Satellitenbildern mit Hilfe von KNNs geht. Allerdings wurde das Problem behandelt, ob auf dem Bild ein Wasserkörper zu sehen ist oder nicht, und

nicht welche Pixel genau zu besagtem Wasserkörper gehören. Ihr Klassifikationsverfahren erreicht eine Genauigkeit von über 98%.

Beide oben genannten Arbeiten trainierten ihr KNN mit manuell erstellten Trainingsdaten. Das hat zur Folge, dass die Genauigkeit und Qualität der Trainingsdaten deutlich höher als in automatisch generierten Trainingsdaten sein kann. Allerdings erfordert es in der Regel einen sehr großen Zeitaufwand eine ausreichend große Anzahl an Trainingsdaten per Hand anzufertigen, besonders wenn es um Segmentierung geht.

3 Training des Neuronalen Netzes

Neuronale Netze funktionieren, indem sie zuerst konstruiert und anschließend trainiert werden. Die Konstruktion ist dabei ein einmaliger Prozess, der von der allgemeinen Aufgabe des neuronalen Netzes abhängt. Für diese Arbeit wird das bereits konstruierte DeepLab-ResNet für Tensorflow von Vladimir Nekrasov verwendet. Dabei handelt es sich um eine Tensorflow-Implementierung der von Chen et al.[5][6] entwickelten Architektur.

Die benutzten Trainingsdaten sind von Google-Maps heruntergeladene Bilderpaare, welche mit Hilfe einer Software, dessen Design und Implementierung Teil dieser Arbeit war, für unsere Zwecke bearbeitet und in das passende Format gebracht werden. Die Generierung der Trainingsdaten ist kein komplett automatischer Vorgang, sondern benötigt mehrere Aktionen, die hintereinander ausgeführt werden müssen. Die Aktionen lassen sich aufteilen in die folgende drei Schritte: Erstens soll, mithilfe der Software, eine Batch-Datei generiert und ausgeführt werden, welche Bilder von Google-Maps herunterlädt. Zweitens wird ein Java Programm ausgeführt, welches die Bilderpaare für das Training bearbeitet und versucht Bildern ohne Flüsse zu entfernen. Drittens wird ein Python Skript ausgeführt, welches das zweite Bild der Bilderpaare in das passende Format bringt.

Insgesamt wurden 9.000 Bilderpaare von Google-Maps heruntergeladen, von denen 543 durch den in Abschnitt 3.2 beschriebenen Filter gelassen und mit denen das Training durchgeführt wurde.

3.1 Herunterladen von zufälligen Google-Maps-Bildern

Die Batch-Datei wird mit Hilfe eines Java Programms automatisch generiert und ist eine lange Liste aus Paaren von ”wget” Befehlen, welche als Parameter eine URL entgegen nehmen und falls es sich dabei um ein Bild handelt dieses Bild herunterladen. Die benutzten URLs sind von maps.googleapis.com, welche ebenfalls Parameter entgegen nehmen können um auf einer Website ein Bild von Google-Maps abzubilden, welches nach den angegebenen Parametern formatiert wird. So ist es möglich einen Ausschnitt der Erde als Satellitenbild abzuspeichern und den gleichen Abschnitt auf alle Wasserkörper zu reduzieren und abzuspeichern. Das in Abbildung 3.1 gezeigte Beispiel wurde durch

```
https://maps.googleapis.com/maps/api/staticmap?center=46.995557,18.
973142&zoom=12&size=400x445&scale=1&style=element:labels|visibility:
off&maptype=satellite&&key=AIzaSyDDa3R0JSLS8rMDlSuG9gUJn2EODHUtRys
```

und

```
https://maps.googleapis.com/maps/api/staticmap?center=46.995557,18.
973142&zoom=12&size=400x400&scale=2&style=element:all|visibility:
```

`off&style=feature:water|element:geometry|visibility:on&maptype=roadmap&&key=AIzaSyDDa3R0JSLS8rMDlSuG9gUJn2EODHUtRys` erzeugt.

Die Koordinaten werden zufällig aus den Intervallen 45,82° – 53,33° für den Breitengrad und 5,94° – 29,32° für den Längengrad gewählt, was in etwa Zentraleuropa entspricht. Der Zoom-Parameter wird zufällig als 12 oder 13 gewählt.

(a) Satellitenbild **(b)** Nur die Wasserkörper

Abbildung 3.1: 2 Bilder von Google-Maps mit unterschiedlichen Parametern

3.2 Bearbeitung und Auswahl der Bilder

Bei beiden Bildern werden zuerst die unteren 45 Pixel abgeschnitten um das Google-Logo zu entfernen, womit das linke Bild auch schon fertig bearbeitet ist. Beim rechten Bild wird zunächst überprüft, wie groß der Anteil der Wasserpixel ist. Ist der Anteil größer als 70% oder kleiner als 0,6% wird das Bilderpaar verworfen, da es sich sehr wahrscheinlich nicht um einen Fluss handelt.

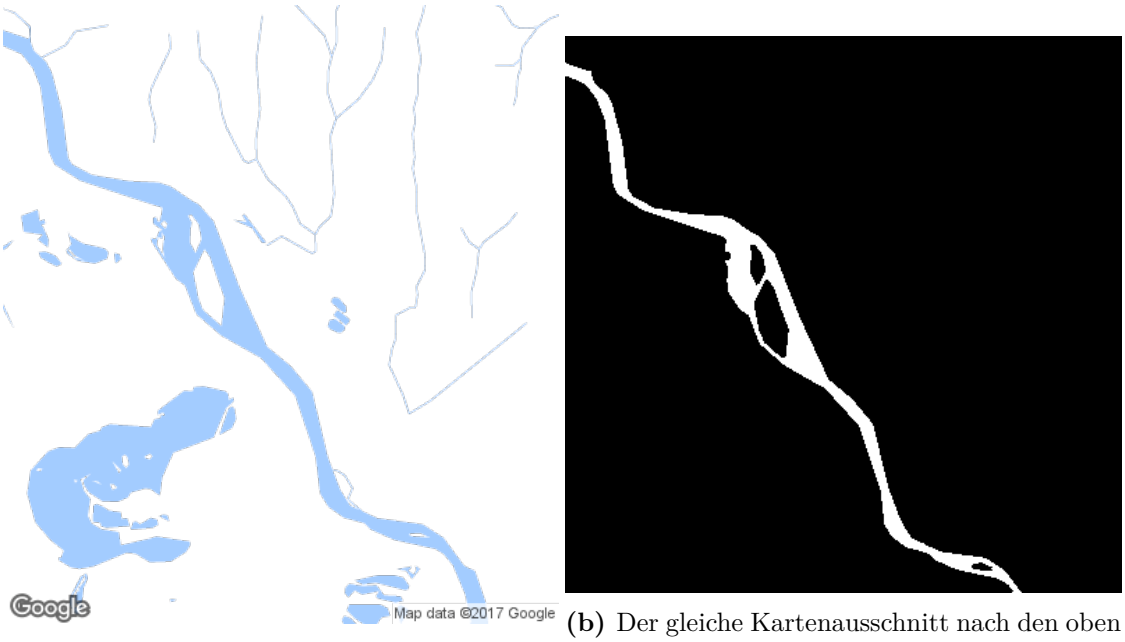
Anschließend wird das Bild binarisiert, sodass Flusspixel weiß werden und der Rest schwarz wird und danach erodiert, sodass weiße Pixel mit weniger als sechs weißen Nachbarpixeln zu schwarzen umgewandelt werden. Dadurch werden häufig auftretende, ein bis zwei Pixel breite Gewässer wie Bäche und Gräben entfernt, die auf dem Satellitenbild nicht sichtbar wären und somit auch nicht im gewünschten Output enthalten sein sollen. Um Rechenzeit zu sparen, werden erneut alle Bilderpaare entfernt, bei denen der Flusspixelanteil größer als 70% oder kleiner als 0,4% ist.

Als letzter Bearbeitungsschritt wird versucht große Seen und kleine stationäre Gewässer zu entfernen, sodass möglichst nur größere Flüsse übrig bleiben, damit das Neuronale

Netz so gut es geht auf Flüsse und nicht auf beliebige Gewässer trainiert wird. Dazu werden die weißen Pixel in ihre Zusammenhangkomponenten zerlegt und beschriftet. Eine Pixelgruppe wurde entfernt, falls sie weniger als 30 Pixel besitzt oder sie alle folgenden Kriterien erfüllt: Sei *height* die Höhe des Bildes, *width* die Breite des Bildes, *regionheight* die Höhe des obersten Pixels der Gruppe minus die Höhe des untersten Pixels der Gruppe, *regionwidth* die Breite des Pixels ganz rechts der Gruppe minus die Breite des Pixels ganz links der Gruppe, *number* die Anzahl der Pixel in der Gruppe und *size* = *regionheight* * *regionwidth*. Dann muss gelten:

$$number \ / \ size < 0{,}2$$
$$5 > regionwidth \ / \ regionheight > 0{,}2$$
$$width \ / \ regionwidth > 2$$
$$height \ / \ regionheight > 2$$

Sind alle Kriterien erfüllt, ähnelt die Pixelgruppe einem See oder stationären Gewässer und wird ebenfalls entfernt. Abbildung 3.2 zeigt ein Beispielbild vor und nach der Bearbeitung.



(a) Eine Karte wie sie im vorherigen Schritt heruntergeladen wurde

(b) Der gleiche Kartenausschnitt nach den oben beschriebenen Bearbeitungen

Abbildung 3.2: Verarbeitungsschritt 2

3.3 Formatierung der Bilder

Damit die Bilder zum Training des KNN verwendet werden können, darf das zweite Bild aus den Bilderpaaren nicht aus drei-dimensionalen RGB Farben, sondern nur aus Farben

aus einem eindimensionalen Farbvektor, den Labels, bestehen. Dazu wird jeder Farbe ein Label von 0 bis 255 zugeordnet. In diesem Falle werden schwarzen Pixeln (0, 0, 0) das Label 0 und weißen Pixeln (255, 255, 255) das Label 1 zugeordnet. Mit Hilfe dieser Labels kann das Bild nun in ein Grauton-Bild konvertiert werden, welches aus bis zu 256 verschiedenen Grautönen bestehen kann. In diesem Format kann das Bild für das Training verwendet werden. Abbildung 3.3 zeigt das fertig verarbeitete Bild, wobei man sehr genau hinsehen muss um den Fluss im rechten Bild wieder zu erkennen.

(a) Die Karte aus dem vorherigen Schritt **(b)** Die gleiche Karte als greyscale-image

Abbildung 3.3: Vorher schwarze Pixel bleiben schwarz, die weißen Pixel werden zu (1, 1, 1) und sind somit fast schwarz und nur sehr schwer zu erkennen.

3.4 Trainingsparameter

Zur Optimierung des Trainings lassen sich sieben verschiedene Parameter variieren:

number of steps	wie viele Iterationen durchlaufen werden
learning rate	wie stark sich die Gewichte pro Iteration anpassen
power	verringert die *learning rate* mit jeder Iteration
weight decay	sorgt dafür, dass Gewichte im Laufe der Zeit gegen 0 gehen
momentum	verstärkt Gewichtsänderungen in die gleiche Richtung wie frühere Änderungen
input size	wie groß das eingelesene Bild ist
batch size	wie viele Bilder pro Iteration benutzt werden

Für *number of steps* gilt in der Regel je mehr desto besser. Auch nachdem jedes Bild einmal eingelesen wurde (eine Epoche), führen weitere Durchläufe zu Verbesserungen. Das einzige Problem, das bei einer geringen Anzahl von Trainingsdaten auftreten kann. ist das sogenannte "over-fitting". Gibt man dem Netzwerk zu wenige Trainingsbilder und diese werden zu oft zum Training verwendet, kann es geschehen, dass die Bilder auswendig gelernt werden. In dem Falle reichen die Parameter im Netzwerk aus, um sich für jedes Eingangsbild das perfekte Ausgangsbild zu merken. Dann spielen die Features und Muster von Flüssen keine Rolle mehr und Bilder, die nicht zum Training benutzt wurden und somit auch nicht auswendig gelernt wurden, können nicht erfolgreich verarbeitet werden. Werden ausreichend viele verschiedene Trainingsbilder benutzt, entfällt diese Gefahr und *number of steps* kann nach Rechenleistung und Zeit gewählt werden. Im Falle dieser Arbeit wurden insgesamt 140.000 Iterationen für das Training durchlaufen, wofür ungefähr 40 Stunden Rechenaufwand, mit einer 6GB Nvidia GTX1070, benötigt wurden.

Für die Ermittlung der *learning rate* existiert kein festes Verfahren und sie wird in der Regel durch Ausprobieren und Testen gewählt. Das KNN benutzt das Gradientenverfahren um die optimalen Gewichtungen ω zu ermitteln. Dafür werden die partiellen Ableitungen der Fehlerfunktion (Gleichung 2.2) für alle Gewichtungen ω_{ij} gebildet.

$$\frac{\partial E}{\partial \omega_{ij}} = \frac{\partial(\frac{1}{2}\|\vec{y} - \vec{y'}\|^2)}{\partial \omega_{ij}}$$

Dabei sind die Neuronen k aus $\vec{y'}$ relevant, die entweder das Gewicht ω_{ij} direkt in ihrem Eingang liegen haben ($j = k$) oder das Gewicht ω_{ij} an einem Neuron anliegt, das durch Rückwärtspropagation von Neuron k aus erreichbar ist. Abbildung 3.4 veranschaulicht, welche Neuronen zur Menge k gehören.

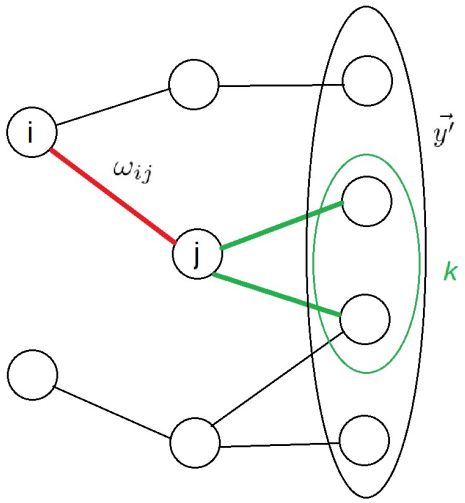


Abbildung 3.4: Schwarz umkreist: Alle Neuronen die zur Menge $\vec{y'}$ gehören.
Grün umkreist: Alle Neuronen k, die abhängig vom Gewicht ω_{ij} sind

Die resultierenden Terme $\partial E/\partial \omega_{ij}$ lassen sich zum Vektor $\partial \vec{E}/\partial \omega$ zusammenfassen, wobei jede Richtung für ein Gewicht steht. Dabei gibt $\partial \vec{E}/\partial \omega$ das Verhältnis der Gewichte an, in dem diese verändert werden müssen um den Fehler E in diesem Punkt mit maximaler Geschwindigkeit wachsen zu lassen. Um den Fehler zu minimieren, wird das Inverse von $\partial \vec{E}/\partial \omega$ gebildet und mit einer Konstanten η multipliziert. Diese Konstante η ist die *learning rate* und gibt im Endeffekt die Schrittgröße an, mit der sich in jeder Iteration die Gewichte verändern.

$$\Delta \omega_{ij} = -\eta \frac{\partial E}{\partial \omega_{ij}} \tag{3.1}$$

Ist die *learning rate* zu groß, kann das Minimum nie genau erreicht werden und im Extremfall der Fehler sogar divergieren. Ist sie zu klein, besteht zum einen die Gefahr in lokalen Minima hängen zu bleiben. Zum anderen werden mehr Iterationen benötigt um das globale Minimum zu erreichen.

Um eine sinnvolle *learning rate* zu finden, wurden vor dem tatsächlichen Training kleinere Datensätze erstellt und diese mit vergleichsweise wenigen Iterationen trainiert und getestet. Dabei wurden alle Parameter bis auf die *learning rate* konstant gehalten.

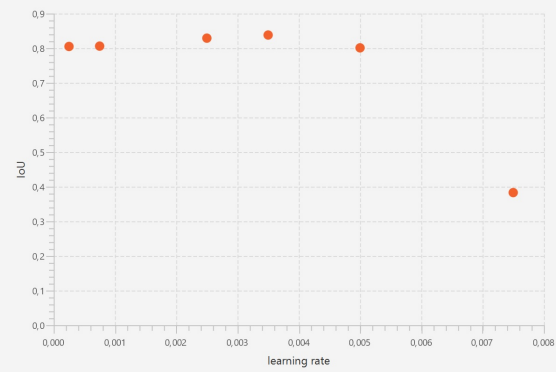


Abbildung 3.5: Die durchschnittliche korrekte Überdeckung zwischen gewünschtem und tatsächlichem Output (Intersection over Union = IoU) in Abhängigkeit von der *learning rate*.

Das beste Ergebnis lieferte eine *learning rate* von 0,0035, welche somit als Anfangswert für die *learning rate* gewählt wurde. Es ist schwer einzuschätzen wie passend dieser Wert ist, da jeder Datensatz und somit die jeweils optimale *learning rate* anders ist. Allerdings stürzte die Genauigkeit in den Testmodellen stark ab, als die *learning rate* zu hoch gewählt wurde (siehe Abbildung 3.5). Im nächsten Absatz wird beschrieben, dass die *learning rate* während des Trainings noch reduziert wird, von daher wird eine hohe Anfangslernrate, die nicht zu Beginn zu Divergenz führt, eine vernünftige Wahl sein. Diese Bedingungen erfüllt die *learning rate* von 0.0035.

Die oben genannten Probleme, die mit einer zu großen oder zu kleinen *learning rate* auftreten, können umgangen werden, falls die *learning rate* mit einem großen Wert anfängt und mit jeder Iteration kleiner wird. Mit einer großen Anfangsrate können lokale Minima umgangen werden und Iterationen gespart werden. Dadurch, dass die *learning rate* am Ende klein ist, wird das globale Minimum genauer angenähert und eine Divergenz wird verhindert. Um diesen Effekt zu erzielen, wird der Parameter $power = p$ eingeführt. Sei lr die effektive Lernrate mit der trainiert wird, $base$ die Anfangslernrate = *learning rate*, $step$ die aktuelle Iteration und max die Anzahl aller geplanten Iterationen = *number of steps*. Dann berechnet sich die effektive Lernrate durch

$$lr = base \cdot \left(1 - \frac{step}{max}\right)^{p}$$

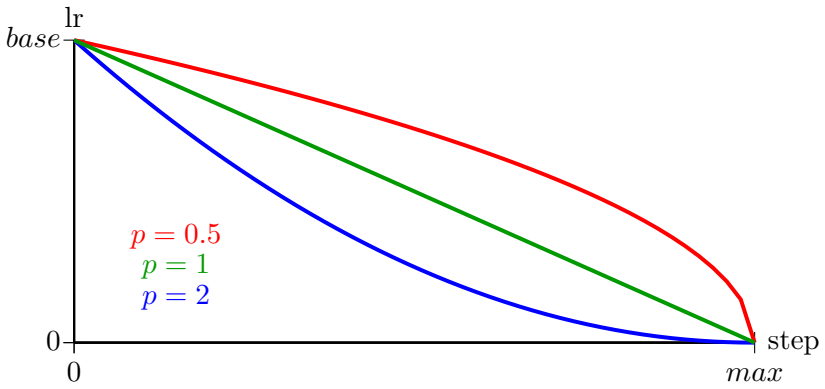


Abbildung 3.6: Verlauf der effektiven Lernrate in Abhängigkeit von verschiedenen p Werten.

Um die Lernrate nicht zu stark abzudämpfen und die letzten Iterationen nicht unbedeutend zu machen, wurde mit 0.9 ein Wert von *power* leicht unter 1 gewählt.

Damit die einzelnen Gewichte nicht über alle Grenzen wachsen können, wird ein Regularisierungsterm in der Form des *weight decays* eingeführt. Der Term $\Delta\omega_{ij}$ aus Gleichung 3.1 wird um den Term $-\eta\lambda\omega_{ij}$ zu

$$\Delta\omega_{ij} = -\eta\frac{\partial E}{\partial\omega_{ij}} - \eta\lambda\omega_{ij}$$

erweitert. Dabei ist λ der *weight decay*. Je größer λ desto mehr werden große Gewichte "bestraft". Um einen guten *weight decay* experimentativ zu ermitteln, wurden ähnlich wie bei der *learning rate* kleinere Datensätze mit verschiedenen *weight decays* traininert um einen möglichst passenden Wert für diese Aufgabenstellung zu finden. Allerdings lieferten alle ausprobierten *weight decays* (von 0.005 bis 0.00005) identische IoUs von 0.884. Das lässt vermuten, dass in diesem konkreten Fall zu groß werdende Gewichte kein großes Problem bieten und der *weight decay* keine wichtige Rolle spielt. Als Trainingswert wurde daher der voreingestellte Standardwert von $\lambda = 0.0005$ gewählt.

Das bei der *learning rate* angesprochene Problem der lokalen Minima wird neben dem Parameter *power* auch durch den Parameter *momentum* bekämpft. Dieser bestimmt wie stark der Einfluss ist, den die letzte Gewichtsänderung $\Delta\omega_{ij}^{alt}$ auf die aktuelle Gewichtsänderung $\Delta\omega_{ij}^{neu}$ nimmt. Sei α das *momentum*, dann gilt

$$\Delta\omega_{ij}^{neu} = \alpha \cdot \Delta\omega_{ij}^{alt} + (-\eta\frac{\partial E}{\partial\omega_{ij}} - \eta\lambda\omega_{ij})$$

und

$$\omega_{ij}^{neu} = \omega_{ij}^{\text{alt}} + \Delta\omega_{ij}^{neu}$$

. Dadurch hat nicht nur die letzte Gewichtsänderung Einfluss auf die aktuelle Gewichtsänderung, sondern rekursiv auch alle vorherigen. Wäre $\alpha = 1$, so würde jede vorherige Gewichtsänderung bei jeder neuen Iteration noch einmal stattfinden. Ist $0 < \alpha < 1$, so haben zwar alle vorherigen Gewichtsänderungen einen Einfluss auf die aktuelle, dieser Einfluss nimmt aber exponentiell ab und nur die unmittelbar vergangenen sind noch relevant. In dieser Arbeit wurde ein Wert von 0.9 für α gewählt.

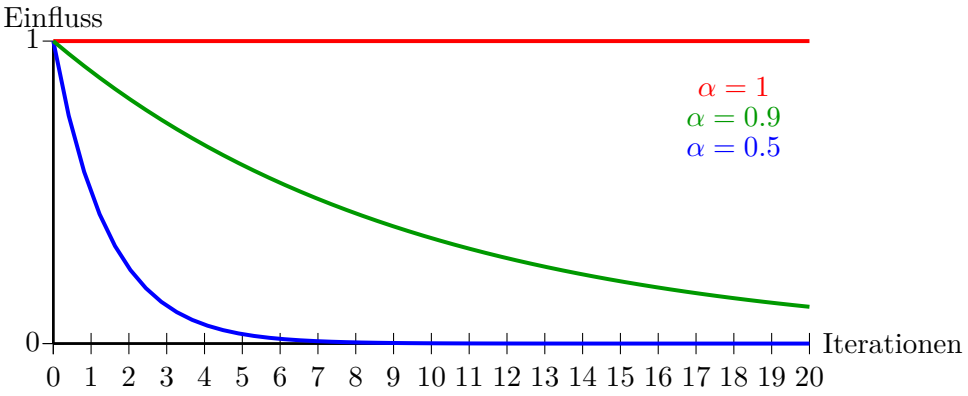


Abbildung 3.7: Einfluss der vergangenen Gewichtsänderungen im Vergleich zur Aktuellen, in Abhängigkeit von der Anzahl der Iterationen, die die vergangene Änderung zurück liegt.

Nicht nur lokale Minima werden durch das *momentum* vermieden, sondern auch Plateaus und Ebenen mit geringen Steigungen im Gradientenfeld $E(\omega)$ werden durch das *momentum* schneller durchquert. Wäre die aktuelle Steigung der einzige Faktor, würde das Gradientenverfahren in Ebenen und Plateaus stark abbremsen und viele Iterationen benötigen um wieder Fortschritt zu machen.

Die *input size* ist die Anzahl der Pixel, die vom Eingangsbildes verwendet werden. Ist die *input size* kleiner als die Ausmaße des Eingangsbildes, wird ein zufälliger Ausschnitt dieser Größe gewählt. Sie hängt stark von der Größe der Eingangsbilder ab und sollte in der Regel genauso groß sein. Ist dies der Fall, führen große Eingangsbilder zu einem genaueren Training, da Features und Facetten der Objekte besser dargestellt werden. Allerdings wächst mit wachsender Eingangsbildgröße der Rechenaufwand und der benötigte Speicherplatz der Grafikkarte. In diesem Falle wurde auf einer Nvidia GTX1070 mit 6GB trainiert, womit das obere Limit für die *input size* in etwa bei 400x400 Pixeln lag und verwendet wurde.

Als letzten Parameter gilt es die *batch size* zu bestimmen. Sie gibt an, wie viele Bilder pro Iteration für das Training verwendet werden. Bei ihr gilt ähnlich wie bei der *number of steps*, je größer desto besser, da ein größerer Wert zu genaueren Gewichtsanpassungen führt und das Training beschleunigt. Allerdings ist sie wie die *input size* durch die Kapazität der Grafikkarte beschränkt. Soll die *batch size* vergrößert werden, muss in der Regel die *input size* verkleinert werden oder ein ressourcenreicheres System benutzt werden.

Da eine größere *batch size* in erster Linie zu einem schnelleren Training führt, wurde in dieser Arbeit der Trade-off zwischen *batch size* und *input size* zugunsten der *input size* entschieden. Das bedeutet, die *input size* wurde mit 400x400 maximal groß gewählt, wodurch die *batch size* auf maximal 3 begrenzt wurde, welcher Wert auch gewählt wurde.

4 Ergebnisse

Um ein quantifizierbares Gütemaß für das entstandene Modell zu haben, wird die mittlere *Intersection over Union* (IoU) gebildet. Dafür wird ein Validierungsdatensatz benötigt, der ähnlich wie ein Trainingsdatensatz aus Paaren von Eingangs- und gewünschten Ausgangsbildern besteht. Für jedes der Eingangsbilder generiert das Modell das Ausgangsbild, welches dann mit dem gewünschten Ausgangsbild aus dem Validierungsdatensatz verglichen wird. Die *Intersection over Union* ist dabei der Anteil der korrekt vorhergesagten Outputs. In diesem Falle entspricht das der Anzahl der identischen Pixel im tatsächlichen und gewünschten Ausgangsbild geteilt durch die Anzahl aller Pixel in einem der beiden Bilder. Für die mittlere *Intersection over Union* wird der durchschnittliche Wert aller Bilder im Validierungsdatensatz gebildet. Sei n die Anzahl an Bilderpaaren im Validierungsdatensatz, a_i die Anzahl der korrekt vorhergesagten Pixel für das i-te Bilderpaar und b_i die Anzahl der falsch vorhergesagten Pixel für das i-te Bilderpaar. Dann ist die mittlere *Intersection over Union*

$$IoU = \frac{1}{n} \sum_{i=1}^{n} \frac{a_i}{a_i + b_i}$$

Das Modell wurde auf drei verschiedene Datensätze angewandt um die Performance in verschiedenen Situationen zu testen. Bei den ersten beiden Datensätzen handelt es sich um Satellitenbilder aus Google-Maps, genauso wie bei den Bildern die zum Training verwendet wurden. Der Unterschied zwischen den beiden Datensätzen besteht darin, dass der zweite aus Bildern mit höherer Auflösung als die Trainingsbilder besteht. Beide Datensätze konnten durch die in Kapitel 3 beschriebenen Schritte automatisch generiert werden.

Beim dritten Datensatz handelt es sich um Fotos, die aus einem Flugzeug aufgenommen wurden. Bei diesen Bildern ließ sich über das in Kapitel 3 beschriebene Verfahren leider kein gewünschter Output generieren, weswegen es sich nur um Eingangsbilder handelt. Ohne die gewünschten Outputs kann zwar keine IoU gebildet werden, allerdings wird in Kapitel 4.3 begründet, warum das in diesem Falle kein großer Verlust ist.

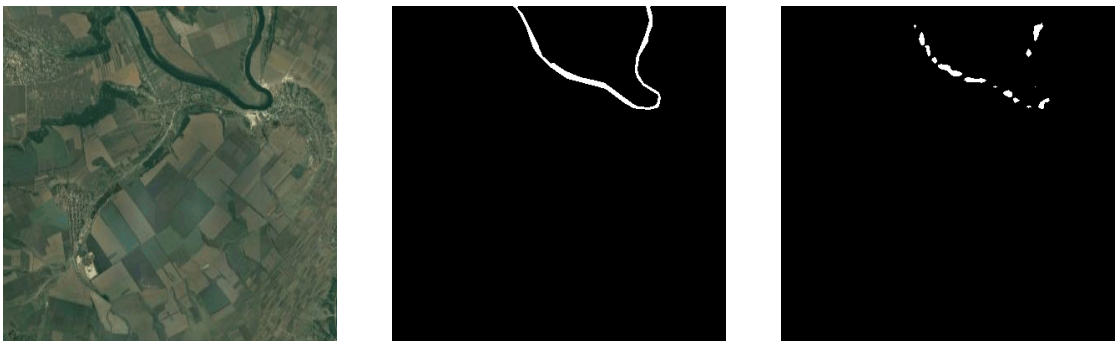
4.1 Satellitenbilder gleichen Formates

Bilder, wie sie zum Training benutzt wurden, werden wie erwartet am besten vom Modell verarbeitet. Der Validierungsdatensatz, bestehend aus 132 Bilderpaaren, liefert eine IoU von 84%. Benutzt man den Trainingsdatensatz als Validierungsdatensatz, wird sogar eine IoU von 92,5% erzielt. Ein Wert von 84% bedeutet, dass das Modell den Fluss auf vielen, aber nicht auf allen Bildern erkennt. Die Bilder, auf denen der Fluss erfolgreich erkannt

wurde, sind die, auf denen der Fluss deutlich zu sehen ist und sich vom Hintergrund abhebt. Die Bilder, bei denen das Modell versagt, enthalten in der Regel sehr schmale und schwer erkennbare Flüsse. Abbildungen 4.1 – 4.3 zeigen drei unterschiedlich erfolgreiche Eingangsbilder.

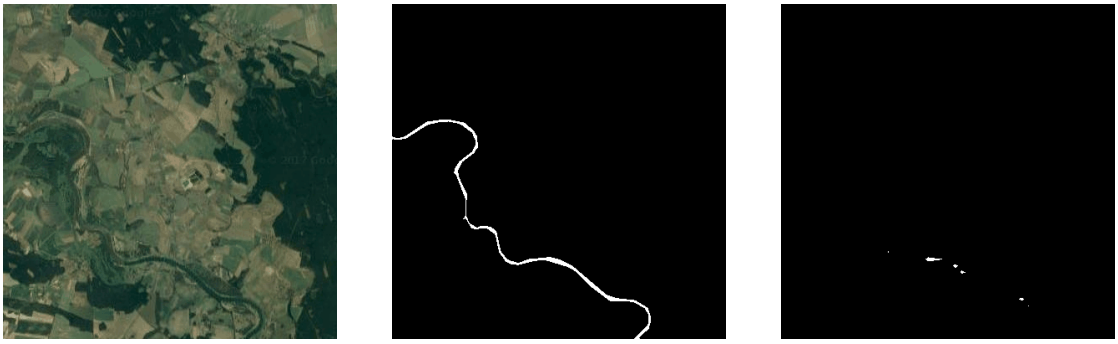
(a) Eingangsbild **(b)** gewünschter Output **(c)** tatsächlicher Output

Abbildung 4.1: Ein gut erkannter Fluss auf einem 400x400 Bild.



(a) Eingangsbild **(b)** gewünschter Output **(c)** tatsächlicher Output

Abbildung 4.2: Ein fast erkannter Fluss auf einem 400x400 Bild.



(a) Eingangsbild **(b)** gewünschter Output **(c)** tatsächlicher Output

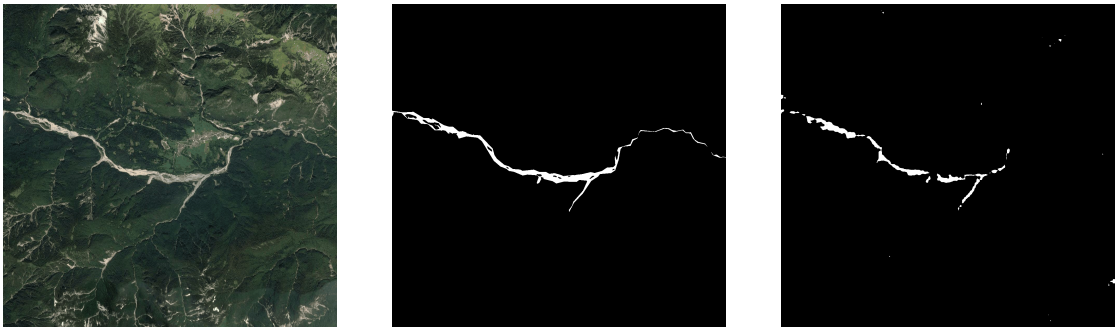
Abbildung 4.3: Ein nicht erkannter Fluss auf einem 400x400 Bild.

4.2 Satellitenbilder mit höherer Auflösung

Erhöht man die Auflösung der Eingangsbilder, fällt die IoU des Modelles. Die Bilder des zweiten Validierungsdatensatz wurden identisch zu den Bildern aus dem ersten Datensatz erzeugt, mit dem einzigen Unterschied, dass sie eine Auflösung von 1200x1200 anstatt 400x400 haben. Der Datensatz besteht aus 242 Bilderpaaren und liefert nur noch eine IoU von 70,4%. Ähnlich wie bei dem ersten Datensatz werden Bilder mit deutlich erkennbaren Flüssen wesentlich besser verarbeitet als die mit dünnen, versteckten Flüssen. Abbildungen 4.4 - 4.6 zeigen drei unterschiedlich erfolgreiche Eingangsbilder.

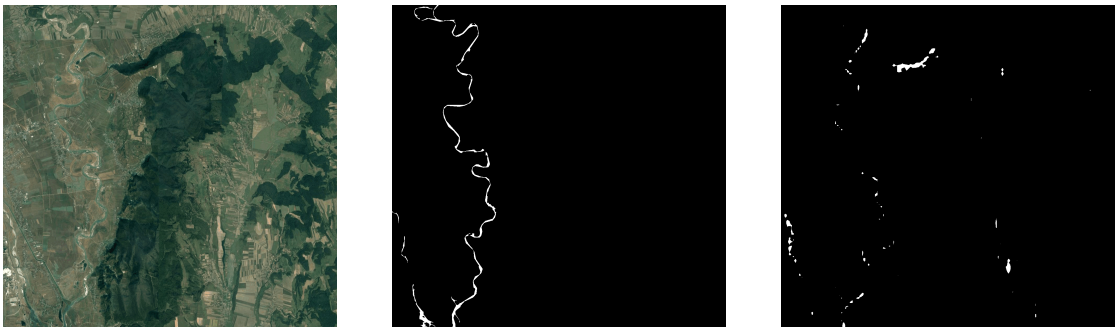
(a) Eingangsbild **(b)** gewünschter Output **(c)** tatsächlicher Output

Abbildung 4.4: Ein gut erkannter Fluss auf einem 1200x1200 Bild.



(a) Eingangsbild **(b)** gewünschter Output **(c)** tatsächlicher Output

Abbildung 4.5: Ein fast erkannter Fluss auf einem 1200x1200 Bild.

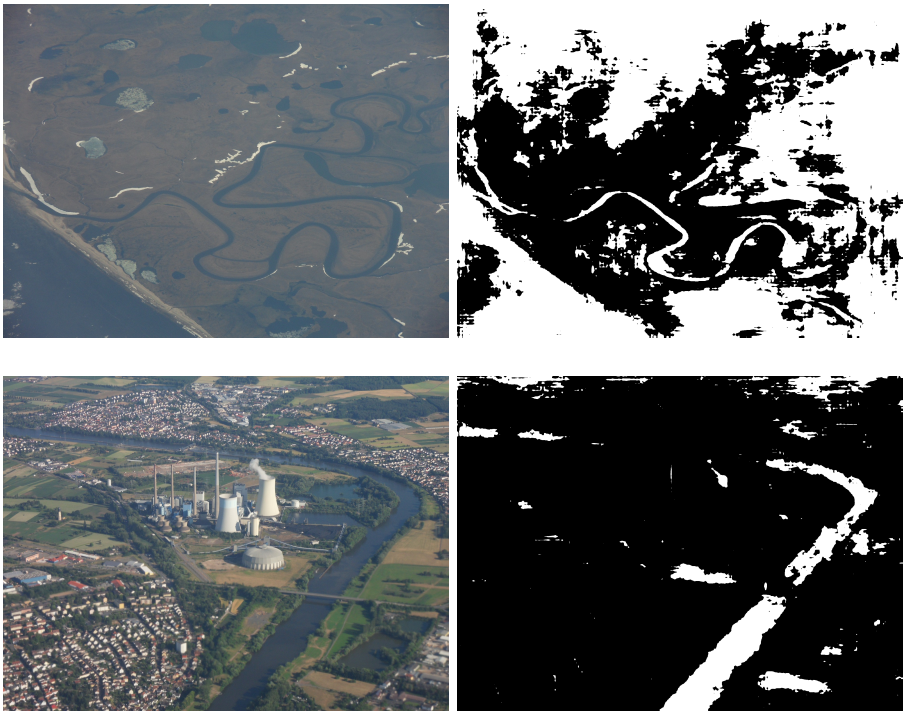


(a) Eingangsbild **(b)** gewünschter Output **(c)** tatsächlicher Output

Abbildung 4.6: Ein nicht erkannter Fluss auf einem 1200x1200 Bild.

4.3 Fotos aus Flugzeugen

Die Fotos aus dem Flugzeug unterscheiden sich erheblich von den Satellitenaufnahmen der ersten beiden Validierungsdatensätze. Zum einen blockieren Störobjekte wie Wolken oder Teile des Flugzeugs die Sicht auf Teile von Flüssen, zum anderen geben Schatten, variabler Lichteinfall und ein geneigter Aufnahmewinkel dem Bild eine ganz andere Struktur als Satellitenbildern. Eine IoU kann zwar nicht gebildet werden, wäre allerdings auch überflüssig, da das Ergebnis ein eindeutiger Fehlschlag ist. Abbildung 4.7 zeigt die beiden dabei noch besten Ergebnisse.



(a) Eingangsbilder **(b)** entsprechende Outputs

Abbildung 4.7: Die beiden besten Bilderpaare aus dem dritten Datensatz. Fotos wurden aus dem Flugzeug geschossen.

5 Diskussion der Ergebnisse

Die Ergebnisse in Tabelle 5.1 zeigen, dass eine präzise Flusserkennung auf einer Luftaufnahme nicht ohne weiteres mit der oben beschriebenen Methode durchführbar ist. Dabei lassen sich drei verschiedene Quellen unterschieden, die für die Ungenauigkeiten verantwortlich sind. Das erste und offensichtlichste Problem besteht darin, dass das KNN nur die Inputs sauber verwerten kann, auf die es auch trainiert wurde. Um eine halb-automatische Methode zum Generieren der Trainingsdaten zu entwickeln, wurden ausschließlich Satellitenaufnahmen von Google-Maps verwendet. Die Menge aller Luftaufnahmen enthält aber auch noch andere Bilder, wie zum Beispiel Fotos aus einem Flugzeug. Diese unterscheiden sich erheblich von den Satellitenbildern von Google-Maps, da sie zum einen nicht im 90 ° Winkel zur Erdoberfläche geschossen werden können, und zum anderen Störobjekte wie Wolken im Weg sein können.

Damit Fotos aus Flugzeugen ebenfalls verarbeitet werden können, müssten auch solche als Trainingsdaten verwendet werden. Allerdings lassen diese sich nicht automatisch generieren und müssten einzeln per Hand erstellt werden. Das manuelle Erstellen von Trainingsbildern ist in der Regel der aufwendigste Schritt beim Konstruieren und Trainieren eines KNNs und wurde in dieser Arbeit versucht zu umgehen.

Datensatz	Training	400x400	1200x1200	Flugzeugfotos
IoU	92,5%	84%	70,4%	n.a.

Tabelle 5.1: Mittlere *Intersection over Union* der verschiedenen Validierungsdatensätze.

Das zweite Problem ist die hohe Leistungsanforderung, die ein KNN an einen Rechner stellt. Ein privater Rechner, wie er für diese Arbeit benutzt wurde, hat nicht die Möglichkeiten das Potential eines KNNs voll auszuschöpfen, da es in der Regel an der Grafikkarte scheitert. Benutzt wurde eine 6GB Nvidia GTX1070, wodurch die *input size* auf 400x400 limitiert war und in einer Stunde ungefähr 3.500 Iterationen berechnet werden konnten. Es ist verständlich, dass ein KNN, das auf 400x400 Bilder trainiert wurde, ungenau wird, sobald es 1200x1200 oder sogar 2800x2800 Bilder verarbeiten soll. Große Auflösungen ermöglichen es sowohl Details, als auch den Zusammenhang von größeren Features besser darzustellen. Genauso wie es für Menschen auf einem verpixelten oder zu stark zugeschnittenen Bild schwerer wird, die abgebildeten Objekte zu identifizieren, kann auch ein KNN größere Bilder verlässlicher charakterisieren. Mit einem spezifisch auf KNN ausgelegten Rechner-Netz können Bilder mit wesentlich größerer Auflösung trainiert werden. Dabei ist zu beachten, dass ein solches Netzwerk nur während des

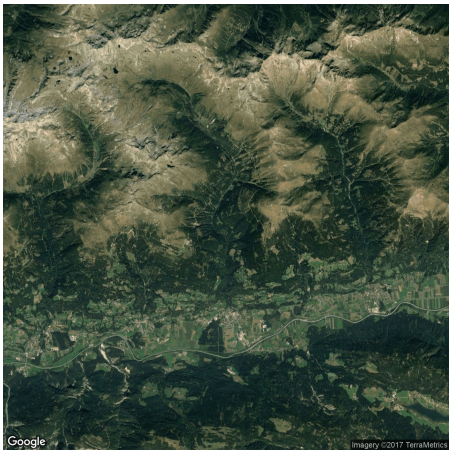
Trainings benötigt wird. Ist das Training abgeschlossen und das Modell erstellt, kann es auch auf deutlich schwächeren Rechnern genutzt werden um eine Inferenz zu einem Eingangsbild zu erstellen.

Das dritte und wahrscheinlich größte Problem ist die Aufgabenstellung selbst. Es ist wahrscheinlich schlichtweg zu schwer ein KNN so zu trainieren, dass es Flüsse und nichts anderes als Flüsse auf allen möglichen Luftaufnahmen erkennt. Dafür können Flüsse zu sehr Straßen oder stationären Wasserkörpern ähneln und Wasserkörper im allgemeinen noch nicht einmal von Menschen zuverlässig identifiziert werden können, da sie auf Luftaufnahmen die gleichen Features haben können wie andere Untergrunde. Abbildung 5.1 zeigt ein Beispiel, bei dem ein See nur schwer als solcher zu erkennen ist. Abbildung 5.2 demonstriert, wie sehr sich eine Straße und ein Fluss ähneln können. Prinzipiell scheint es so zu sein, dass ein KNN die Flüsse gut erkennt, die auch Menschen auf den ersten Blick bemerken.

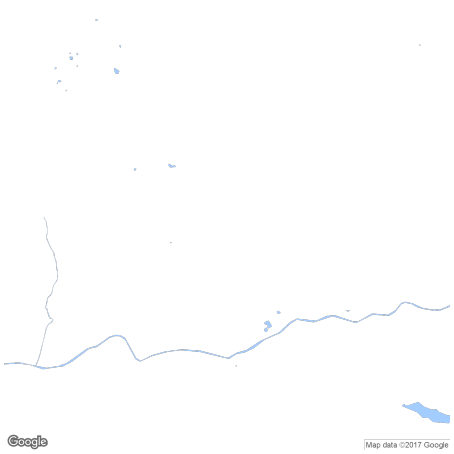
Die in Kapitel 2.2 vorgestellten Arbeiten waren in der Lage, eine deutlich höhere IoU zu erreichen, allerdings lässt sich das durch die Breite der Aufgabenstellung erklären. Je konkreter und spezialisierter eine Aufgabenstellung ist, desto besser ist ein KNN in der Lage ein passendes Modell zu erstellen, da es weniger Variablen zu berücksichtigen gibt. Das spiegelt sich in den Ergebnissen der drei verschiedenen Testsätze wieder. Je mehr Annahmen als gegeben angesehen werden können (Größe des Bilder muss 400x400 sein, es muss sich um ein Satellitenbild handeln und der Fluss muss groß und deutlich zu erkennen sein) desto besser kann das Modell seine Aufgabe erfüllen. Die Aufgabe auf einer beliebigen Luftaufnahme einen beliebigen Fluss zu erkennen, ist, wie bereits oben erklärt, eine deutlich komplexere Aufgabenstellung als die Fokussierung auf einen einzigen Fluss oder eine binäre Ausgabe.

(a) Ein Google-Maps Bild **(b)** Alle Wasserkörper im linken Bild

Abbildung 5.1: Der untere See ist auf dem linken Bild nicht von Feldern und Wäldern zu unterscheiden.

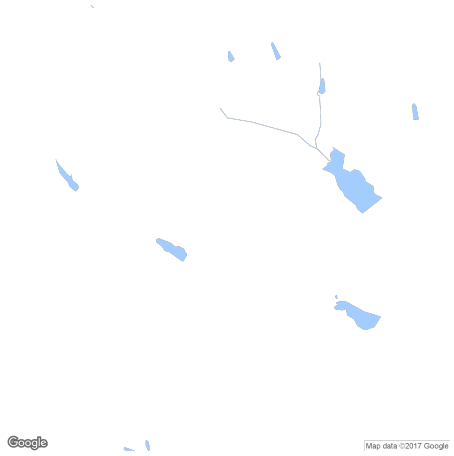


(a) Eingangsbild mit Fluss



(b) Alle Wasserkörper im linken Bild

(c) Eingangsbild ohne Fluss



(d) Alle Wasserkörper im linken Bild

Abbildung 5.2: Bei diesem Maßstab ist ein Fluss von einer Straße nicht zu unterscheiden

Es stellt sich die Frage, ob ein KNN überhaupt der richtige Ansatz zum Lösen dieser Aufgabenstellung ist. Die oben genannten Probleme lassen vermuten, dass andere Verfahren einem KNN für diese spezielle Aufgabe überlegen sind. Dabei muss meiner Meinung nach unterschieden werden, was genau erreicht werden soll. Wenn das einzige Ziel darin besteht Flüsse auf Flugzeug-Fotos zu erkennen, kann ich mir gut vorstellen, dass man effiziente Algorithmen dafür entwickeln kann. Allerdings werden diese Algorithmen nur auf diese eine Art von Fotos anwendbar sein, genauso wie das in dieser Arbeit trainierte KNN nur auf Satellitenfotos anwendbar ist. Da das Training eines KNN sehr große Rechenleistung benötigt, und in dieser Arbeit nicht das volle Potential ausgeschöpft werden konnte, sind wahrscheinlich anderen Ansätze effektiver um Flüsse auf Flugzeug-Fotos zu erkennen.

Ist das Ziel allerdings die Erkennung von Flüssen auf jeder Art von Luftaufnahme,

so wie es zu Beginn dieser Arbeit verlangt wurde, könnten KNNs den einzig validen Lösungsansatz bieten. Dafür müssten allerdings deutlich mehr Aufwand in das Training gesteckt werden, und es müssten viele Trainingsbilder per Hand erstellt werden. Zum Vergleich: Der *PASCAL VOC 2012* Datensatz, der zum Testen verschiedenster KNNs verwendet wird, besteht aus 10.582 Bildern alleine für das Training [7]. Mit dieser Anzahl von Bildern, die alle möglichen Arten von Luftaufnahmen und nicht nur Satellitenbilder beinhalten und einem spezifisch für KNN aufgebauten Rechnernetzwerk, sehe ich für Neuronale Netze kein Problem darin Flüsse auf Luftaufnahmen zu erkennen.

6 Zusammenfassung und Ausblick

Das Ziel der Arbeit, Flüsse automatisch auf Luftaufnahmen zu erkennen, um sie dann für andere Anwendungen nutzen zu können, wurde im strengen Sinne nicht erreicht. Satellitenbilder können zwar erfolgreich verarbeitet werden, allerdings kann das entstandene Modell Fotos aus Flugzeugen nicht verwerten, was allerdings eine Voraussetzung für die anfangs erwähnte Anwendung der Lokalisierung ohne GPS wäre. Diese Arbeit war der Versuch den normalerweise aufwändigsten Arbeitsschritt beim Training eines KNNs, namentlich das Erstellen der Trainingsdaten, zu automatisieren. Allerdings beschränkt diese Automatisierung den Erfolg des Modells auf Satellitenbilder. Um die oben erwähnte Anwendung doch realisieren zu können, könnte der nächste Versuch das Training eines neuronalen Netzes mit manuell erstellten Trainingsdaten sein. Die Tests mit Satellitenbildern haben allerdings gezeigt, dass das Modell die Bilder erfolgreich verarbeiten kann, auf denen der Fluss auch für Menschen sehr deutlich sichtbar ist. Für das manuelle Training und die spätere Anwendung wäre es demnach wahrscheinlich sinnvoll nur die Bilder auszuwählen, auf denen ein Fluss unverwechselbar und eindeutig abgebildet ist.

Eine andere Verbesserung wäre das Nutzen eines Systems mit mehr Ressourcen. In den Tests hat sich gezeigt, dass eine größere Auflösung der Trainingsbilder hilfreich wäre um Satellitenbilder besser verarbeiten zu können, allerdings wird ein mächtigeres System nicht das Problem mit Fotos aus Flugzeugen lösen.

Literaturverzeichnis

[1] J. Long, E. Shelhamer, and T. Darrell, "Fully convolutional networks for semantic segmentation," in *The IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, June 2015.

[2] C. M. Bishop, *Neural Networks for Pattern Recognition* -. Oxford: Clarendon Press, 1995.

[3] M. R. Casado, R. B. Gonzalez, T. Kriechbaumer, and A. Veal, "Automated identification of river hydromorphological features using uav high resolution aerial imagery," *Sensors*, vol. 15, no. 11, pp. 27969–27989, 2015.

[4] A. K. Goswami, S. Gakhar, and H. Kaur, "Article: Automatic object recognition from satellite images using artificial neural network," *International Journal of Computer Applications*, vol. 95, pp. 33–39, June 2014. Full text available.

[5] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Semantic image segmentation with deep convolutional nets and fully connected crfs," *CoRR*, vol. abs/1412.7062, 2014.

[6] L. Chen, G. Papandreou, I. Kokkinos, K. Murphy, and A. L. Yuille, "Deeplab: Semantic image segmentation with deep convolutional nets, atrous convolution, and fully connected crfs," *CoRR*, vol. abs/1606.00915, 2016.

[7] M. Everingham, L. Van Gool, C. K. I. Williams, J. Winn, and A. Zisserman, "The PASCAL Visual Object Classes Challenge 2012 (VOC2012) Results." http://www.pascal-network.org/challenges/VOC/voc2012/workshop/index.html.

Erklärung

Hiermit versichere ich die vorliegende Abschlussarbeit selbstständig verfasst zu haben, keine anderen als die angegebenen Quellen und Hilfsmittel benutzt zu haben, und die Arbeit bisher oder gleichzeitig keiner anderen Prüfungsbehörde unter Erlangung eines akademischen Grades vorgelegt zu haben.

Würzburg, den 28. August 2017

.....
Jonas Jung