

ATTACK FACE OBFUSCATION WITH DEEP LEARNING USING CONVOLUTION NEURAL NETWORKS

Antonio Galeazzi (inf102867@fh-wedel.de)
und
Till Hildebrandt (inf102835@fh-wedel.de)

29. März 2018

INHALTSVERZEICHNIS

1	Einleitung	3
1.1	Weichzeichnen	4
1.2	Verpixelung	4
2	Datenquellen und Aufbereitung	5
3	Neuronale Netze	8
3.1	Perzeptron	8
3.2	Sigmoid-Neuronen	10
3.3	Architektur neuronaler Netze	11
3.4	Lernen - Das Anpassen der Gewichte	12
3.5	Gradientenabstieg (Gradient Descent)	13
3.6	Convolutional Networks	15
4	Frameworks	20
4.1	Scikit-Learn	21
4.2	Keras	21
4.3	TensorFlow	22
5	Bericht, Meilensteine und Ziele des Projekts	24
5.1	Verschaffen eines Überblicks über den State of the Art	26
5.2	Beschaffung von entsprechenden Daten und de- ren Aufbereitung	26

5.3	Erlangung von theoretischen Kenntnissen	27
5.4	Modellierung eines Convolutional Networks	27
5.5	Umsetzung des zuvor definierten Modells in TensorFlow	27
5.6	Trainieren des Netzes	28
5.7	Produzieren von brauchbaren Ergebnissen	28
6	Fazit	30
6.1	Trainingsdatensätze	30
6.2	Lerndauer	30

1 EINLEITUNG

Machine Learning stellt einen Aspekt der künstlichen Intelligenz dar, der in der vergangenen Zeit an immer größerer Bedeutung gewonnen hat. In diesem Kontext ist insbesondere das Deep Learning hervorzuheben, das wiederum einen Teilbereich des Machine Learnings darstellt. Dessen Popularität lässt sich zum Einen damit erklären, dass es die Geschwindigkeit und Reife heutiger Prozessoren (CPU/GPU/TPU¹/FPGA²) zulässt Ergebnisse in akzeptabler Zeit zu erzielen und zum Anderen damit, dass durch das stetige Anwachsen der durchs Internet erzeugten Daten, genug Material zur Verfügung steht, mit dem gearbeitet werden kann. Besonders im Kontext von Bilderkennungen und Klassifizierungsproblemen sind Techniken des Machine Learnings kaum noch wegzudenken.

In bildgebenden Medien, Videos wie Fotos, werden Gesichter von Menschen verfälscht, um deren Identität unkenntlich zu machen.³ Diese Technologien werden von öffentlichen Medien, wie Privatpersonen verwendet. In der Vergangenheit gab es den Fall eines Kinderschänders, der verfälschte Gesichtsbilder von sich veröffentlichte. Er verwendete dabei ein Verfahren, das Pixel um einen zentralen Punkt zu einer Spirale rotiert. Behörden war es damals möglich, diese Form der Gesichtsverfälschung, der Informationsverlust im Vergleich zu den Verfahren, die in dieser Arbeit behandelt werden, gering ist, aufzuheben und das Gesicht weitgehend wiederherzustellen.⁴ Motiviert unter anderem dadurch, stellt diese Arbeit eine Grundlagenanalyse dar, in wie weit CNNs dafür verwendet werden können sehr viel verbreitetere, aber destruktive Obfuscation-Verfahren anzugreifen.

Maßgeblich kommen beim Verfälschen zwei Verfahren zum Einsatz³: “Weichzeichnen” (Gaussian Blur)⁵ und “Verpixelung” (Pixelization)⁶.

¹ Wikipedia, Tensor Processing Unit.
(https://de.wikipedia.org/wiki/Tensor_Processing_Unit)

² Wikipedia, Field Programmable Gate Array.
(https://de.wikipedia.org/wiki/Field_Programmable_Gate_Array)

³ Andrew Senior, Protecting Privacy in Video Surveillance, S 130 ff.

⁴ Wikipedia, “Christopher Paul Neil”.
(https://en.wikipedia.org/wiki/Christopher_Paul_Neil)

⁵ Wikipedia, Gaussian Blur.
(https://en.wikipedia.org/wiki/Gaussian_blur)

⁶ Wikipedia, Pixelization.
(<https://en.wikipedia.org/wiki/Pixelization>)

1.1 Weichzeichnen

Der gaußscher Weichzeichner oder Gaussian smoothing, beschreibt ein Verfahren, mit dem der Kontrast von Bildern verringert wird. Damit wird der Verlust von Detailinformationen erreicht. Die mathematische Formel, nach der die Transformation funktioniert, lautet:

$$G(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{x^2}{2\sigma^2}}$$

Abbildung 1: Gleichung der gausschen Weichzeichnung zweier Dimensionen.

x und y beschreiben die Distanz zum Ursprung der jeweiligen Achse, σ ist ein Parameter der Funktion, der beschreibt wie sehr die Weichzeichnung streut (siehe [Abbildung 2](#)). Der Formel kann man entnehmen, dass die Farbinformationen benachbarter Pixel in das Ergebnis des aktuell zu berechnenden Pixels miteinfließen. Hier werden die Informationen verschiedener Pixel auf den selben Wertebereich eines Pixels abgebildet. Der dadurch entstehende Informationsverlust ist irreversibel.



Abbildung 2: Vergleichsbild Weichzeichnen mit verschiedenen Parametern.

1.2 Verpixelung

Auch Mosaic-Verfahren, meint eine Menge an Verfahren, die die Auflösung von Bildern oder Bereiche derer künstlich verringern, um Detailinformationen zu verbergen. Hierfür wird

der unkenntlich zu machende Bereich in gleichmäßige Unterbereiche aufgeteilt und deren resultierender Farbwert aus den Pixeln des Ursprungsbildes gemittelt. Bei dieser Verfahrensfamilie gibt es eine Vielzahl an Variationen, die sich in Größe und Form der Unterbereiche und dem genauen Algorithmus, der verwendet wird, um die Unterbereiche unkenntlich zu machen.

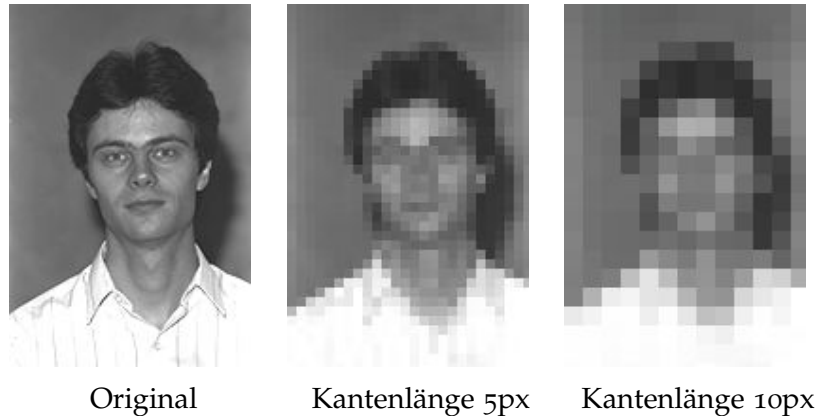


Abbildung 3: Vergleichsbild Weichzeichnen mit verschiedenen Kantenlängen.

Der in diesem Verfahren betrachtete Parameter (siehe Abbildung 3), entspricht der Kantenlänge der resultierenden verpixelten Unterbereiche.

2 DATENQUELLEN UND AUFBEREITUNG

Als grundlegende Datenquelle wurden die *color FERET Database*⁷, die von dem National Institute of Standards and Technology veröffentlicht wurde, und die Datenbank *FaceScrub* von *vintage*⁸ verwendet.

Die *color FERET Database*⁷ umfasst 11.338 Gesichtsbilder von 1.208 Menschen, die in einer kontrollierten Umgebung aufgenommen worden sind, und hält neben Metadaten über Pose,

⁷ NIST, color FERET Database.

(<https://www.nist.gov/itl/iad/image-group/color-feret-database>)

⁸ vision & interaction groupe, FaceScrub.

(<http://vintage.winklerbros.net/facescrub.html>)

Geschlecht, Ethnie und Alter auch weiterführende Daten bereit wie Augenposition und Kamerawinkel. Die Daten liegen im Portable Pixmap Format RGB- und im Graustufenformat homogen in einer vor Maximalauflösung von 512×768 Pixeln vor.

Die *FaceScrub*⁸ Datenbank umfasst über 100.000 Gesichtsbilder von 530 prominenten Menschen, die in einer unkontrollierten Umgebung aufgenommen worden sind und stellt keine weiteren Metadaten bereit. Die Daten liegen im JPEG-Format in einer vor Maximalauflösung von 512×768 Pixeln vor.

Um trotz der vergleichsweise geringen Datenmenge. Vergleichbare Projekte⁹ verwenden hingegen 60.000 bis 300.000 Bilder, interpretierbare Ergebnisse erzielen zu können, beschränkt sich diese Arbeit auf die Verwendung möglichst homogener Bilder unterschiedlicher Personen. Von besonderem Interesse ist hierbei die Pose des Abgebildeten. Die Datenbank unterscheidet Frontal- und Profilbilder sowie Bilder, in denen der Kopf um einen bestimmten Winkel gedreht ist. Als grundlegenden Datensatz wurde sich für die Frontalbilder entschieden, da diese mit 2.722 Bilder von 994 Personen den größten Teildatensatz ausmachen.

Die benötigten Testdatensätze wurde mithilfe von ImageMagick¹⁰ in Version 6.8. aufbereitet. Um die Komplexität der Problemstellung weiter zu reduzieren, wurden die Bilder grauskaliert und auf 12,5% der Ursprungsgröße skaliert, sodass die Trainingsdaten noch eine Auflösung von 64×96 Pixeln haben. Es wurden vier unterschiedliche Testdatensätze mit folgenden CLI-Befehlen generiert¹¹:

⁹ Richard McPherson, Rezar Shokri, Vitali Shmatikov, Defeating Image Obfuscation with Deep Learning.

(<https://arxiv.org/pdf/1609.00408.pdf>)

¹⁰ ImageMagick Studio LLC, ImageMagick.

(<https://www.imagemagick.org/>)

¹¹ Das folgende BASH-Skript `scripts/create_images.sh` erzeugt die Testdaten. Notwendig hierfür sind die Pakete *imagemagick* und *imagemagick-doc*.

Code-Auszug 1: convert - Synopsis

```
convert [input-options] input-file [output-options] output-file
```

Code-Auszug 2: Testdatenerstellung - Graustufen

```
#!/bin/bash
```

```
# every call scales the input image down to 12.5% of its
# original size and grayscales it.
```

```
# convert test data: gaussian-blur (sigma = 3)
```

```
convert <input_file.ppm> \
  -set colorspace Gray \    # grayscale durch
  -separate \                # separates (RGB)
  -average \                 # Mittel
  -scale 12.5\% \           # Skalierung auf 96px x 64px
  -gaussian-blur 0x3 \       # blur
  <output_file.pgm>; mv <output_file.pgm> <output_file.ppm>
```

```
# convert test data: gaussian-blur (sigma = 6)
```

```
convert <input_file.ppm> \
  -set colorspace Gray \
  -separate \
  -average \
  -scale 12.5\% \
  -gaussian-blur 0x6 \
  <output_file.pgm>; mv <output_file.pgm> <output_file.ppm>
```

```
# convert test data: pixelization (edge length = 5px)
```

```
convert <input_file.ppm> \
  -set colorspace Gray \
  -separate \
  -average \
  -scale 12.5\% \
  -scale $(( bc <<< "scale=100;100/5" ))\% \
  -scale 500\% \
  <output_file.pgm>; mv <output_file.pgm> <output_file.ppm>
```

```
# convert test data: pixelization (edge length = 10px)
```

```
convert <input_file.ppm> \
  -set colorspace Gray \
  -separate \
  -average \
  -scale 12.5\% \
  -scale $(( bc <<< "scale=100;100/10" ))\% \
  -scale 1000\% \
  <output_file.pgm>; mv <output_file.pgm> <output_file.ppm>
```

3 NEURONALE NETZE

Die nachfolgenden Ausführungen und Graphiken zum Schaffen eines grundlegenden Verständnisses des Machine Learnings basieren im wesentlichen auf dem E-Book “Neural Networks and Deep Learning”¹².

Im Rahmen des maschinellen Lernens stellen die neuronalen Netze einen elementaren Ansatz dar, der in vielen weiteren Modellen Verwendung findet. Neuronale Netze wie das maschinelle Lernen an sich stellen eine andere Herangehensweise dar, als die klassischer, deterministischer Algorithmen.

Anstatt dem System eine eindeutige Abfolge von Anweisungen mitzuteilen, um eine konkrete Problemstellung zu lösen, definiert man ein Modell und konfrontiert dieses mit verschiedenen Beispielen - die Beispiele sind dabei Tupel aus Eingangsgröße und erwarteter Ausgangsgröße. Die Dimensionen von Eingangs- und Ausgangsgröße können sich dabei gleichen, müssen es aber nicht. So können als Eingabe Bilder dienen und als Ausgaben konkrete Klassen, um beispielsweise Hunde von Katzen unterscheiden zu können.

Anstatt nun algorithmisch zu definieren, was einen Hund von einer Katze unterscheidet, überlässt man es dem zuvor erstellten Modell anhand der gegebenen Eingaben und erwarteten Ausgaben, eigenständig Regeln abzuleiten, um mit dessen Hilfe auch unbekannte Eingaben klassifizieren zu können.

Dieser Ansatz wird als *Soft Computing* bezeichnet.

3.1 Perzeptron

Als elementaren Bestandteil eines neuronalen Netzes dient das *Perzeptron* - dieses stellt die kleinste Einheit eines neuronalen Netzes dar und wird auch als “künstliches Neuron” bezeichnet.

¹²Michael Nielsen, Neural Networks and Deep Learning.
(<http://neuralnetworksanddeeplearning.com/index.html>)

Grundsätzlich akzeptiert ein Neuron einen beliebig großen Input bestehend aus Features x_1, x_2, \dots, x_n und berechnet daraus ein Ergebnis.

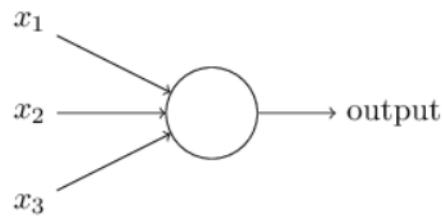


Abbildung 4: Perzeptron

Im gezeigten Bild ist beispielsweise ein Neuron dargestellt, das drei Inputgrößen akzeptiert und daraus einen Output produziert. Um den Output zu berechnen werden Gewichte (*engl. weights*) eingeführt. Ob das Neuron 0 oder 1 als Output liefert, hängt dann davon ab, ob die gewichtete Summe der Eingangsgrößen einen zu definierenden Schwellwert überschreitet.

Dies kann anhand des nachfolgenden Bilds verdeutlicht werden:

$$\text{output} = \begin{cases} 0 & \text{if } \sum \omega_i x_i \leq \text{threshold (Schwellwert)} \\ 1 & \text{if } \sum \omega_i x_i > \text{threshold (Schwellwert)} \end{cases}$$

Abbildung 5: Berechnung des Outputs.

Dies ist das grundlegende Modell. Grundsätzlich kann man sich das Perzeptron als einen “Entscheidungs-Unterstützer” vorstellen, der eine Entscheidung trifft, in dem er konkrete Fakten mit einem bestimmten Gewicht versieht.

Das gezeigte Modell ist augenscheinlich sehr simpel und noch sehr weit von dem entfernt, was man als ein neuronales Netz bezeichnen würde. Es ist allerdings ohne Weiteres denkbar, das gezeigte Modell komplexer zu gestalten, indem mehrere Perzeptrons miteinander verknüpft werden, so dass beispielsweise das nachfolgende Netzwerk entstehen könnte:

In Grafik⁵ wurde ein Schwellwert eingeführt, der überschritten werden muss, damit ein Perzeptron aktiviert wird. Um das Modell zu vereinheitlichen, kann der *Bias* definiert werden, der

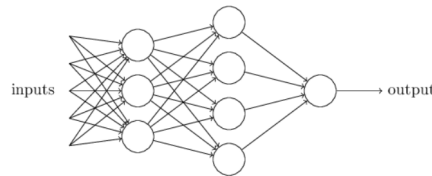


Abbildung 6: Mehrschichtiges neuronales Netz.

den negativen Schwellwert darstellt. Durch diese Maßnahme kann die Aktivierungsfunktion des Perzeptrons dann geschrieben werden als:

$$\text{output} = \begin{cases} 0 & \text{if } \sum \omega \cdot x + b \leq 0 \\ 1 & \text{if } \sum \omega \cdot x + b > 0 \end{cases}$$

Abbildung 7: Berechnung des Outputs bei Verwendung eines Bias.

Inhaltlich kann das Bias als ein Maß verstanden werden, aus dem hervorgeht, wie leicht ein Perzeptron aktiviert werden kann. Nimmt der Bias einen großen Wert an, so kann das Perzeptron einen Wert von 1 annehmen, auch wenn das Produkt aus den Gewichten und den Eingangsgrößen einen negativen Wert annimmt. Gleiches gilt selbstverständlich auch für einen kleinen Bias, der zur Folge hat, dass ein Perzeptron träger reagiert.

3.2 Sigmoid-Neuronen

Eine Weiterentwicklung des zuvor vorgestellten Modells stellen Sigmoid-Neuronen dar. Diese Weiterentwicklung wird dann erforderlich, wenn das Anpassen der Gewichte - also letztlich das Lernen - betrachtet wird. Dabei ist das Ziel, dass eine kleine Anpassung eines Gewichts auch nur eine kleine Änderung des Outputs zur Folge hat. Das zuvor betrachtete Perzeptron ist lediglich in der Lage 0 oder 1 als Output zu liefern, so dass Änderungen an den Gewichten keine stetige Änderung des Outputs zur Folge haben, sondern folgenlos bleiben können bis irgendwann ein Sprung von 0 auf 1 oder umgekehrt stattfindet, was wiederum eine große Änderung darstellt.

Die Weiterentwicklung besteht nun in einer Verfeinerung der Aktivierungsfunktion. Anstatt eine Sprungfunktion^{9b} zu verwenden, die lediglich 0 und 1 als Funktionswert annehmen kann, wird die Sigmoid Funktion^{9a} eingeführt, die die folgende Form hat:

$$\sigma(z) \equiv \frac{1}{1 + e^{-z}}$$

Abbildung 8: Sigmoid-Funktion.

Der entscheidende Unterschied kann an den beiden nachfolgenden Grafiken verdeutlicht werden, die jeweils die Kurve der entsprechenden Funktion darstellen:

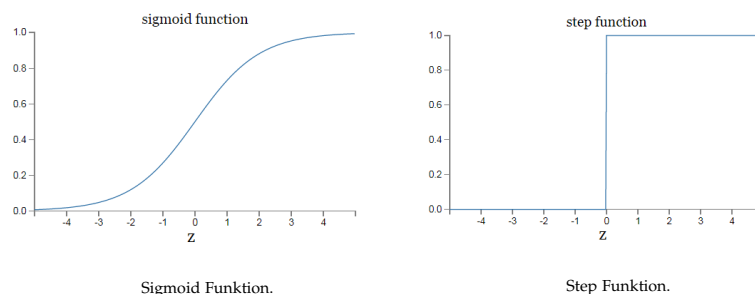


Abbildung 9: Vergleich der Aktivierungsfunktionen *Sigmoid* und *Step-Funktion*

3.3 Architektur neuronaler Netze

Mit diesen Bestandteilen als Ausgangspunkt können nun tatsächlich konkretere Neuronale Netze und deren Architekturen eingeführt werden. Neuronale Netze bestehen üblicherweise aus mehreren Schichten, den *Layern*. Diese lassen sich grundsätzlich in drei Kategorien aufteilen: Input, Hidden und Output. Neuronale Netze beinhalten für gewöhnlich ein Input-Layer und ein Output-Layer sowie dazwischen beliebig viele Hidden-Layer. Die Form der Input- und Output-Layer ist dabei sehr naheliegend: das Input-Layer hat die gleiche Struktur wie die des Inputs und das Output-Layer hat entsprechend die gleiche Struktur wie der Output.

Angenommen es sollen Bilder der Größe 28×28 Pixel klassifiziert werden und es gibt 10 mögliche Klassen, dann besteht das Input-Layer aus $28 \times 28 = 784$ Neuronen und das Output-Layer aus 10 Neuronen.

Lediglich der Bereich zwischen Input- und Output-Layer - die Hidden-Layer - lässt sich nicht ohne Weiteres aus dem Input oder dem Output ableiten. Es gibt lediglich Heuristiken, die beim Design der Hidden-Layer angewandt werden können, allerdings keine konkreten Regeln, die befolgt werden müssen. Diese Struktur kann anhand des nachfolgenden Bilds verdeutlicht werden, bei dem - um die Übersichtlichkeit zu wahren - das Input-Layer etwas komprimiert dargestellt wird:

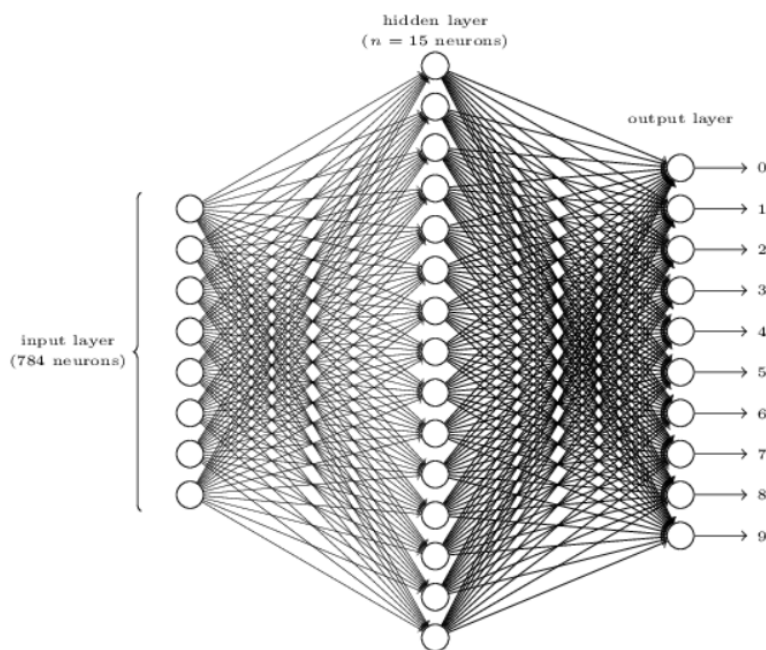


Abbildung 10: Hidden-Layer Darstellung.

3.4 Lernen – Das Anpassen der Gewichte

Das Lernen stellt den zentralen Ansatz von neuronalen Netzen dar. Eng im Zusammenhang mit dem Lernen steht eine Kosten-Funktion, die häufig auch als Verlust-Funktion bezeichnet werden kann. Diese stellt letztlich den Fehler zwischen dem Erwartungswert und dem tatsächlichen Wert, den das neuronale Netz berechnet, dar. Mathematisch betrachtet ist das grundlegende

Prinzip des Lernens diese Funktion zu minimieren, also zu gewährleisten, dass die Abweichungen zwischen Erwartungswert und tatsächlichem Wert möglichst gering sind. Es sind grundsätzlich viele verschiedene Verlust-Funktionen denkbar, eine, die jedoch eine breite Verwendung findet, ist die quadratische Kosten-Funktion - auch als *mean squared error* (MSE) bezeichnet.

$$C(w, b) \equiv \frac{1}{2n} \sum_x \| y(x) - a \|^2$$

Abbildung 11: Mean Squared Error (MSE).

Dabei beschreiben w und b die Gewichte bzw. die Bias des neuronalen Netzes und n stellt die Anzahl der Trainingsdaten dar. Der Vektor a beschreibt den Output des Netzes und $y(x)$ stellt den Erwartungswert zu einem Input x dar. Das Ziel besteht nun darin, die Gewichte und Bias so zu manipulieren, dass die gezeigte Funktion einen möglichst kleinen Wert annimmt.

3.5 Gradientenabstieg (Gradient Descent)

Um die soeben eingeführte Kostenfunktion zu minimieren, wird ein Verfahren verwendet, das als Gradientenabstieg (*engl. gradient descent*) bezeichnet wird. Neben dem hier vorgestellten Verfahren existieren noch viele weitere Methoden, die zur Lösung dieses Optimierungs- bzw. Minimierungsproblems verwendet werden können. Zur Veranschaulichung des Problems soll nun eine Funktion zweier Variablen, die minimiert werden soll, betrachtet werden.

Dieses Problem könnte selbstverständlich durch Verfahren der Analysis gelöst werden. In der Realität ist die Anzahl der Variablen jedoch um ein Vielfaches höher, sodass diese Verfahren nicht mehr effizient angewandt werden können.

Zur Visualisierung der grundlegenden Idee des Gradientenabstiegs kann sich ein Ball vorgestellt werden, der sich abwärts in Richtung eines Tals bewegt. Bewegt man den Ball etwas in

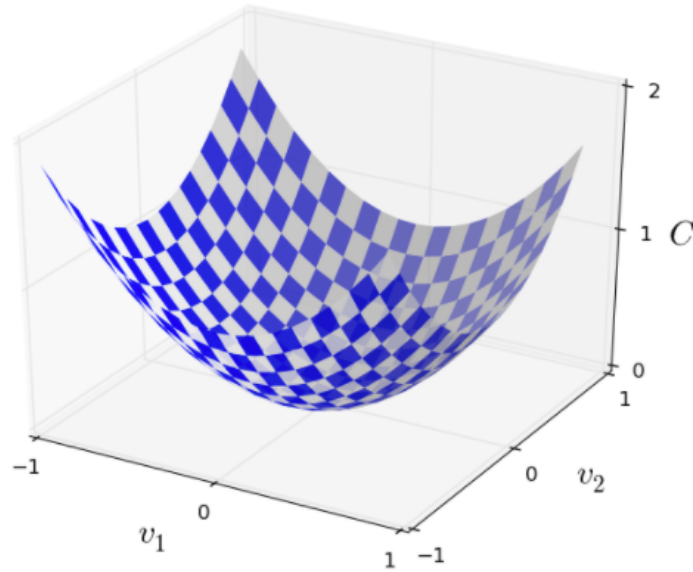


Abbildung 12: Darstellung des Gradientenabstieg bei einer Funktion in Abhängigkeit von zwei Variablen.

Richtung v_1 und etwas in Richtung v_2 so gilt für den Funktionswert C :

$$\Delta C \approx \frac{\delta C}{\delta v_1} \Delta v_1 + \frac{\delta C}{\delta v_2} \Delta v_2$$

Abbildung 13: Veränderung des Funktionswerts.

Der Gradient wird dann folgendermaßen definiert:

$$\nabla C \equiv \left(\frac{\delta C}{\delta v_1}, \frac{\delta C}{\delta v_2} \right)^T$$

Abbildung 14: Definition des Gradienten

Durch die Verwendung dieses mathematischen Objekts kann die vorherige Gleichung¹³ formuliert werden als:

Wird die Veränderung der Variablen (Gewichte) nun so gewählt, dass sie dem Inversen des Gradienten multipliziert mit einem kleinen, positiven Faktor (Lernrate) entspricht, kann sichergestellt, dass sich dem Minimum der Funktion genähert wird.

$$\delta C \approx \nabla C \cdot \delta v$$

Abbildung 15: Veränderung des Funktionswerts durch Gradienten ausgedrückt

Dieses Vorgehen wird mehrfach - genau genommen für jeden Durchlauf der Trainingsdaten - wiederholt, bis die dadurch erzielte Näherung den Ansprüchen an die Genauigkeit genügt.

3.6 Convolutional Networks

Insbesondere bei der Klassifizierung von Bildern hat sich eine bestimmte Art von neuronalen Netzen als besonders passend herausgestellt - dabei handelt es sich um die *Convolutional Networks*. Diese zeichnen sich dadurch aus, dass sie auch die räumliche Struktur der Bilder berücksichtigen: Während herkömmliche *fully connected* neuronale Netze sämtliche Pixel gleich behandeln würden, unabhängig davon, ob sie beispielsweise benachbart sind oder nicht, betrachten Convolutional Networks immer nacheinander konkrete Abschnitte eines Bilds, um Strukturen zwischen benachbarten Pixeln erkennen und somit verarbeiten zu können. Dabei liegen den Convolutional Networks im Wesentlichen drei Ideen zugrunde:

1. Local Receptive Field
2. Shared Weights
3. Pooling

3.6.1 Wesentliche Konzepte von Convolutional Networks

Diese Bestandteile sollen nun nachfolgend einzeln genauer betrachtet werden.

3.6.1.1 Local Receptive Field

Für das Verständnis des lokalen rezeptiven Felds (*Local Receptive Field*) empfiehlt es sich, die Input-Neuronen des Convolutional Networks nicht als vertikale Linie von Neuronen, sondern vielmehr in den Dimensionen des Bildes, das von dem Netzwerk betrachtet werden soll, zu visualisieren. Werden beispielsweise Bilder der Größe 28×28 Pixel betrachtet entspräche das dem nachfolgenden Input-Layer:

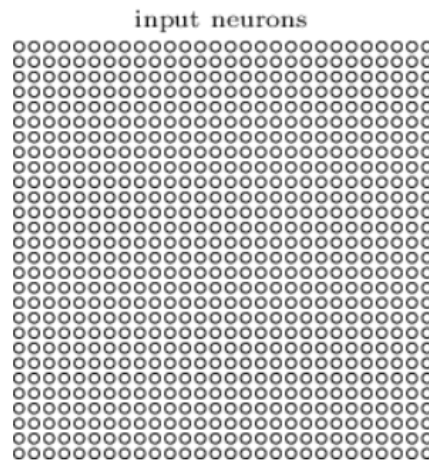


Abbildung 16: Feld von Inputneuronen.

Im Unterschied zu *herkömmlichen* neuronalen Netzen, bei denen üblicherweise alle Input-Neuronen mit allen Neuronen des nachfolgenden Hidden-Layers verbunden werden, besteht die Besonderheit nun darin, dass jedes Neuron des Hidden-Layers mit einem kleinen Bereich des Inputs verbunden wird. Dieses Vorgehen kann anhand des folgenden Bildes veranschaulicht werden:

Dieses lokale rezeptive Feld wird dann gemäß der Konfiguration über das Input-Bild *bewegt*, so dass alle Input-Neuronen besucht werden. Demnach sähe der zweite Schritt - unter der Annahme, dass das Feld immer um ein Pixel verschoben wird - folgendermaßen aus:

Auf diese Art und Weise entsteht das erste Hidden-Layer dessen Größe selbstverständlich etwas kleiner ist, als die Größe des Input-Layers. Geht man von einem Input-Bild der Größe 28×28 Pixel und einem lokalen rezeptiven Feld der Größe 5×5

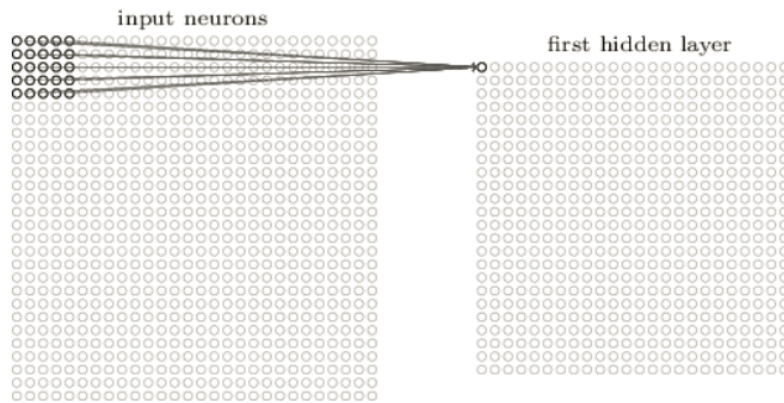


Abbildung 17: Convolution-Prozess, Schritt 1.

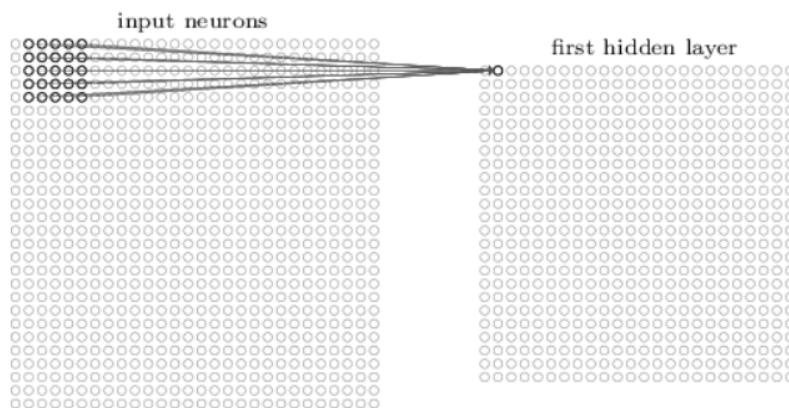


Abbildung 18: Convolution-Prozess, Schritt 2.

Pixel aus, so folgt daraus ein erstes Hidden-Layer der Größe 24×24 Pixel.

3.6.1.2 Shared Weights

Jedes Neuron des Hidden-Layers verfügt wie üblich über einen Bias und - im Falle eines lokalen rezeptiven Felds der Größe 5×5 Pixel - über 5×5 Gewichte. Die Besonderheit besteht nun darin, dass diese Gewichte und Bias für *alle* Neuronen des Hidden-Layers gleich gelten. Inhaltlich hat dies zur Folge, dass ein konkretes Hidden-Layer genau eine konkrete *Auffälligkeit* extrahiert. Solch eine konkrete *Auffälligkeit* könnte beispielsweise eine vertikale oder horizontale Linie sein, die durch die Verwendung des lokalen rezeptiven Felds, das sich über das Bild

bewegt, an beliebigen Positionen aufgespürt werden kann. Üblicherweise reicht es nicht aus, nur eine *Auffälligkeit* aufzuspüren. Aus diesem Grund besteht ein einziges Layer häufig aus mehreren parallelen Feature Maps.

So könnte ein einziges Hidden-Layer beispielsweise die nachfolgende Form haben, durch das dann drei verschiedene Auffälligkeiten extrahiert werden würden:

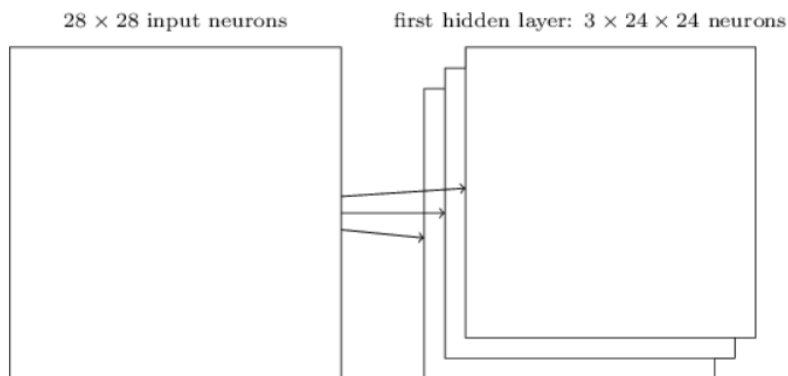


Abbildung 19: Drei Filterschichten.

Ein weiterer wesentlicher Vorteil bei der Verwendung von *Shared Weights* besteht darin, dass die Anzahl der zu trainierenden Variablen um ein Vielfaches verringert wird, wodurch ein schnelleres Lernen ermöglicht werden kann. Eine einzige Feature Map würde bei den bisher betrachteten Dimensionen durch $5 \times 5 = 25$ Gewichte und einen Bias, also insgesamt 26 Parameter definiert werden. Selbst wenn ein Hidden-Layer aus 20 features maps bestünde, so würde dies im Ergebnis zu *nur* $20 \times 26 = 520$ Parametern führen. Wird ein fully connected Layer bestehend aus 30 Neuronen und das gleiche Input-Layer wie zuvor betrachtet, so folgt daraus, dass insgesamt $(28 \times 28) \times 30 + 30 = 23.550$ Parameter in jedem Schritt des Lernens angepasst werden müssen.

3.6.1.3 Pooling

Die zuvor vorgestellten Layer, die durch die Verwendung des lokalen rezeptiven Felds und gemeinsam geteilter Gewichte entstehen, werden gemeinhin als *convolutional* Layer bezeichnet.

net. Von diesen kann ein weiterer wesentlicher Bestandteil von convolutional networks abgegrenzt werden - die *pooling* Layer. Diese Art von Layer folgt für gewöhnlich auf die zuvor vorgestellten *convolutional* Layer und hat zur Aufgabe, die dadurch gewonnenen Informationen zu vereinfachen. Das grundsätzliche Vorgehen kann durchaus mit dem des convolutional Layers verglichen werden - auch bei den pooling Layers kommt ein Feld zum Einsatz, das sich über das Ergebnis des vorherigen Schrittes bewegt. Es könnte sich beispielsweise um ein Feld der Größe 2×2 handeln. Dieses würde somit immer 4 Neuronen betrachten und - im Falle des max-poolings (einer konkreten Ausprägung des Poolings) - das größte von ihnen auswählen.

Bildlich kann dies auf die folgende Art und Weise veranschaulicht werden:

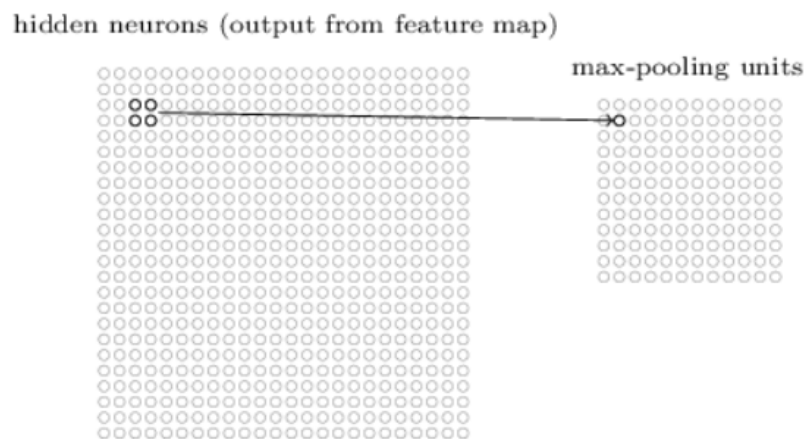


Abbildung 20: Maxpooling Prozess.

Dadurch, dass vier Neuronen auf ein Neuron projiziert werden, verringert sich die Größe bei diesem konkreten Beispiel von 24×24 auf 12×12 . Dieses Vorgehen wird selbstverständlich auf alle feature maps des vorherigen convolutional Layers angewendet, so dass das folgende Ergebnis entsteht.

3.6.2 Beispielhaftes Convolutional Network

Nachdem die wesentlichen Bestandteilen eines convolutional Netzes vorgestellt worden sind, können diese nun miteinander verknüpft werden, um die Architektur eines exemplarischen Netzes zu veranschaulichen. Wie auch bei *normalen* neuronalen

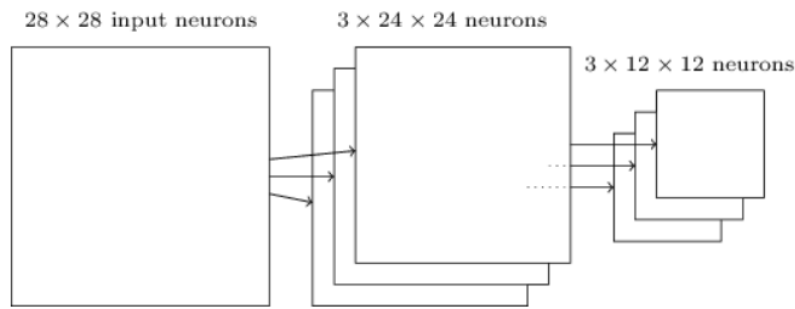


Abbildung 21: Convolution Prozess.

Netzen orientieren sich Input- und Output-Layer an der Struktur des Inputs bzw. Outputs. Dazwischen befinden sich abwechselnd convolutional und pooling Layer, die grundsätzlich beliebig oft hintereinander geschaltet werden können. Demnach sähe ein sehr simples convolutional Netzwerk beispielsweise folgendermaßen aus:

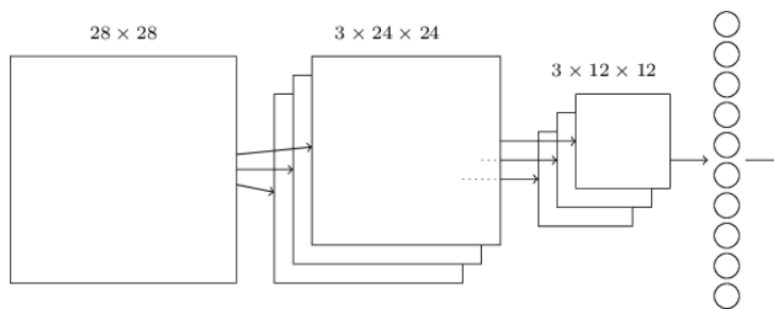


Abbildung 22: Vollständiges *Convolutional Network* mit Klassifizierungs-Ausgabe.

4 FRAMEWORKS

Im Umfeld des Machine Learnings sind zahlreiche Frameworks für verschiedene Plattformen und verschiedene Sprachen zu finden. Diese sind auf unterschiedlichen Abstraktionsebenen angesiedelt und dementsprechend unterschiedlich komplex in der Verwendung. Je höher ein Framework dabei angesiedelt, desto einfacher ist es üblicherweise zu bedienen, bietet dem Anwender aber dafür nur sehr stark eingeschränkte Möglichkeiten. Frameworks hingegen, die auf einem niedrigen Level operieren, bieten dem Anwender sehr detaillierte und komple-

xe Möglichkeiten, den gesamten Programmfluss zu beeinflussen, sind dadurch aber auch deutlich komplexer zu verwenden. Die Frameworks sind insbesondere darauf ausgerichtet, eine Implementation für die konkreten mathematischen Berechnungen und Modelle zu bieten, die darüber hinaus auch hinsichtlich der Performance optimiert worden sind.

Zu nennen wären beispielsweise die nachfolgenden drei Frameworks:

1. Scikit-Learn¹³
2. Keras¹⁴
3. TensorFlow¹⁵

Diese Aufzählung ist nicht vollständig und stellt lediglich einige Repräsentanten dar.

4.1 Scikit-Learn

Scikit-Learn stellt einen simplen Einstieg in die Welt des Machine Learnings dar und hält den Großteil der eigentlichen Komplexität vor dem Anwender verborgen. Scikit-Learn baut im Wesentlichen auf NumPy [<http://www.numpy.org/>] und SciPy [<https://www.scipy.org/>] auf und bietet eine Reihe von einfach zu verwendenden Werkzeugen, die für die Analyse und Auswertung von Daten genutzt werden können.

4.2 Keras

Ähnlich wie Scikit-Learn stellt Keras ebenfalls ein Framework dar, das auf einer verhältnismäßig hohen Ebene operiert. Die

¹³scikit-learn, Machine Learning in Python.
(<http://scikit-learn.org/stable/>)

¹⁴Keras: The Python Deep Learning library.
(<https://keras.io/>)

¹⁵TensorFlowTM, An open-source machine learning framework for everyone.
(<https://www.tensorflow.org/>)

Besonderheit besteht dabei darin, dass Keras eine Schnittstelle zu komplexeren Frameworks wie beispielsweise TensorFlow oder Theano bereitstellt. Damit eignet Keras sich insbesondere für das schnelle und einfache Entwickeln von Prototypen.

4.3 TensorFlow

Von den beiden zuvor genannten Frameworks kann TensorFlow abgegrenzt werden. TensorFlow bietet eine Reihe von verschiedenen komplexen APIs, die dem Anwender Zugang zu verschiedenen Leveln ermöglichen. Die Architektur des Frameworks kann mit Hilfe der nachfolgenden Grafik verdeutlicht werden:

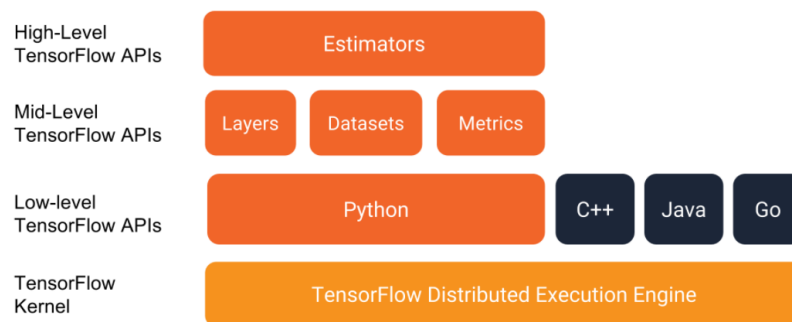


Abbildung 23: Architektur von TensorFlow

Wie Keras und Scikit-Learn bietet auch TensorFlow die Möglichkeit, Verfahren zu verwenden, die leicht zu bedienen sind und somit keine tiefgehenden Kenntnisse voraussetzen. Darüber hinaus ist es allerdings auch möglich, eigene Modelle und Strukturen von Grund auf selbst zu definieren und so direkten Einfluss auf das Lernen, Validieren und Vorhersehen zu nehmen.

4.3.1 Verwendung von TensorFlow

In diesem Projekt wurde der Fokus insbesondere auf TensorFlow gelegt. Den elementaren Datentyp von TensorFlow stellen Tensoren dar. Diese sind mathematisch betrachtet eine Abstraktion von Vektoren und Matritzen. So entspricht ein Tensor des Typs $(0, 0)$ beispielsweise Skalaren und Tensoren vom Typ $(1, 0)$

Vektoren. Im Sinne von TensorFlow wird unter einem Tensor¹⁶ ein Platzhalter für den Output einer Operation¹⁷ verstanden. Die Besonderheit besteht dabei darin, dass ein Tensor lediglich einen Platzhalter darstellt und keine konkreten Werte repräsentiert. Der Tensor repräsentiert vielmehr eine Berechnungsvorschrift, nach der sein Output zu berechnen ist. Auf diese Art und Weise können mehrere Operationen miteinander verkettet werden, die jeweils einen Tensor als Input und Output haben. Dadurch entsteht ein Graph¹⁸, der demnach eine komplette Berechnung repräsentiert. Sobald diese Berechnung im Rahmen einer Session¹⁹ ausgeführt wird, *fließt* der Input durch den zuvor definierten Graphen, so dass schlussendlich ein Output produziert wird.

TensorFlow macht demnach starken Gebrauch des *DataFlow-Paradigmas*²⁰, wodurch insbesondere die Parallelisierung von Berechnungen erheblich begünstigt wird.

Das Zusammenspiel von Graphen und Sessions kann anhand der nachfolgenden Graphik verdeutlicht werden:

¹⁶TensorFlow, tf.Tensor.

(https://www.tensorflow.org/api_docs/python/tf/Tensor)

¹⁷TensorFlow, tf.Operation.

(https://www.tensorflow.org/api_docs/python/tf/Operation)

¹⁸TensorFlow, tf.Graph.

(https://www.tensorflow.org/api_docs/python/tf/Graph)

¹⁹TensorFlow, tf.Session.

(https://www.tensorflow.org/api_docs/python/tf/Session)

²⁰Wikipedia, Dataflow programming.

(https://en.wikipedia.org/wiki/Dataflow_programming)

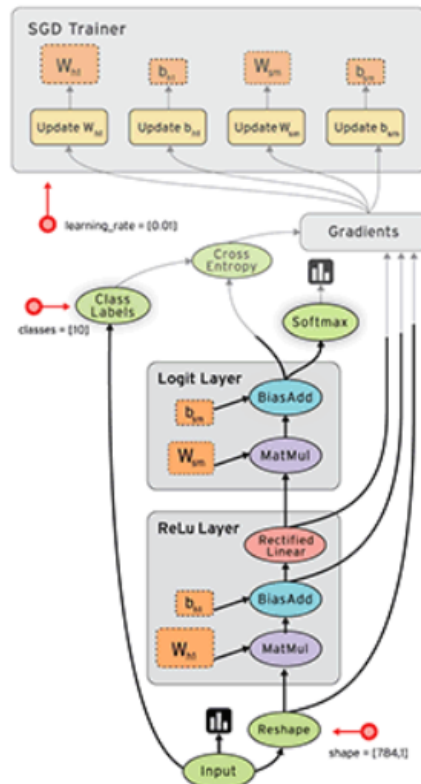


Abbildung 24: Exemplarischer *Dataflow*.

Da im Rahmen des Projekts auf einige Probleme bei der Verwendung von TensorFlow gestoßen wurde, wurde parallel zum Betrieb von TensorFlow versucht, das Modell mit Keras nachzubauen. Keras ist etwas abstrakter als TensorFlow und bietet dem Anwender so weniger Konfigurationsmöglichkeiten, wodurch andererseits auch eigene Fehler ausgeschlossen oder eliminiert werden können.

5 BERICHT, MEILENSTEINE UND ZIELE DES PROJEKTS

Nachfolgend soll dem Leser ein Einblick in den Ablauf des Projekts gewährt werden. Dabei soll auf Meilensteine, die im Vorfeld definiert worden sind, eingegangen werden, Probleme thematisiert werden, die während der Durchführung aufgetreten

sind, Anpassungen und Änderungen, die im Zuge des Auftretens von Problemen vorgenommen werden mussten, dargelegt werden und schlussendlich auch Erkenntnisse, die durch die Durchführung des Projekts gesammelt werden konnten, aufgezeigt werden.

Das thematische Ziel (die möglichst gute Wiederherstellung von unkenntlich gemachten Gesichtern) wurde bereits an anderer Stelle umfangreich thematisiert.

Neben dem inhaltlichen Ziel galt es auch insbesondere tiefgehende Kenntnisse über das Machine Learning zu erlangen. Dabei handelte es sich zum einen um die theoretische Seite, die sich zusammensetzte aus den grundlegenden Prinzipien des Machine Learnings, neuronalen Netzen und ganz konkreten Convolutional Networks, und zum anderen um die eher praktische Seite, die durch die Verwendung des Frameworks TensorFlow charakterisiert wurde.

Im Vorfeld wurden insbesondere die nachfolgenden wesentlichen Meilensteine festgelegt:

1. Verschaffen eines Überblicks über den State of the Art und vergleichbare Projekte und Forschungsarbeiten
2. Beschaffung von entsprechenden Daten und deren Aufbereitung
3. Erlangung von theoretischen Kenntnissen über neuronale Netze (insbesondere Convolutional Networks)
4. Modellierung eines Convolutional Networks
5. Umsetzung des zuvor definierten Modells in TensorFlow (und anderen Frameworks)
6. Trainieren des Netzes
7. Produzieren von brauchbaren Ergebnissen

5.1 Verschaffen eines Überblicks über den State of the Art

Das Thema Machine Learning erfährt derzeit einen unfassbar großen Hype. Infolgedessen finden sich viele verschiedene Forschungsarbeiten und Projekte, die sich mit der Verarbeitung von Bildern beschäftigen. Insbesondere fällt dabei auf, dass die Qualität des Ergebnisses maßgeblich von der Qualität und insbesondere auch der Masse der Trainingsdaten abhängt. Projekte, die sich mit vergleichbaren Thematiken befassen, haben beispielsweise mit mehreren hunderttausend Datensätzen trainiert. Die Anzahl der in diesem Projekt verwendeten Datensätze beläuft sich auf etwa knapp 2.500 Bilder, was offensichtlich eine deutlich kleinere Größenordnung ist. Ein weiterer entscheidender Parameter ist die Anzahl der Epochen. Eine Epoche meint dabei, dass jeder Datensatz der Trainingsdaten einmal betrachtet worden ist. Durch eine Erhöhung der Anzahl der Epochen ist es möglich, das zu trainierende Modell präziser auf die Daten abzustimmen - dabei gilt es jedoch zu beachten, dass zu viele Epochen dazu führen können, dass es zum *Overfitting* kommt und das Modell "zu gut" an die Trainingsdaten angepasst wird, so dass über unbekannte Datensätze deutlich fehlerbehaftete Aussagen getroffen werden können.

5.2 Beschaffung von entsprechenden Daten und deren Aufbereitung

Da die Daten, wie im vorherigen Absatz beschrieben, einen elementaren Anteil zur Qualität des Ergebnisses leisten, galt es im ersten Schritt zunächst, diese zu erlangen und entsprechend aufzubereiten. Unter der Aufbereitung der Daten wird dabei zum einen verstanden, diese zu vereinheitlichen, und zum anderen, sie so anzupassen, dass das Modell mit ihnen effektiv trainiert werden kann. Aus diesem Grund wurde sich beispielsweise dazu entschieden, die Bilder auf eine Größe von 96 x 64 Pixel zu reduzieren und sie auf einen Farbkanal zu reduzieren, so dass Graustufen-Bilder entstehen, um so die Trainingsdauer deutlich verringern zu können.

5.3 Erlangung von theoretischen Kenntnissen

Wie bereits eingangs erwähnt stand neben dem inhaltlichen Ziel auch der Aufbau von neuem Wissen im Vordergrund. Durch die intensive Auseinandersetzung mit verschiedensten Forschungsarbeiten aus dem Bereich des Machine Learnings im Kontext der Verarbeitung von Bildern sowie dem Bearbeiten von Literatur, konnte ein tiefergehendes Verständnis für die grundlegenden Prinzipien des Machine Learnings und insbesondere auch neuronale Netze erlangt werden. Darüber hinaus konnten auch Erfahrungen im Umgang mit weit verbreiteten Frameworks, die im wissenschaftlichen Bereich und in der Bildverarbeitung verwendet werden, gesammelt werden. Dazu zählen insbesondere NumPy²¹. Da im Rahmen der Veranstaltung "Soft Computing" bereits erste Berührungspunkte verzeichnet werden konnten, bestand der Wunsch dies noch zu vertiefen und sich auf einer niedrigeren Ebene mit den Thematiken zu befassen. Aus diesem Grund wurde TensorFlow als Unterstützung gewählt, dass dem Anwender tiefere Einblicke gewährt, als es beispielsweise bei Scikit-Learn der Fall ist.

5.4 Modellierung eines Convolutional Networks

Eine spannende Phase stellte die Modellierung des Convolutional Networks, das für die Verarbeitung der Bilder benutzt werden sollte, dar. Dabei galt es, die theoretischen Kenntnisse, die im Kapitel *Neuronale Netze* genauer beschrieben werden, anzuwenden und durch eine entsprechende Kombination der verschiedenen Arten von Layern ein Modell zu entwickeln, in das die zuvor beschafften Daten eingespeist werden können.

5.5 Umsetzung des zuvor definierten Modells in TensorFlow

Dieser Meilenstein und der zuvor genannte fanden größtenteils parallel statt, da die theoretischen Überlegungen zumeist um-

²¹NumPy, NumPy.
(<http://www.numpy.org/>)

gehend umgesetzt wurden, um deren Praktikabilität beurteilen zu können. Die Umsetzung des Modells stellte dabei nur einen Teil der gesamten Umsetzung dar. Darüber hinaus musste zunächst ein entsprechender Rahmen geschaffen werden, in dem das Modell Verwendung finden konnte. Dies kann anhand des Repositorys nachvollzogen werden.

5.6 Trainieren des Netzes

Einen zentralen Bestandteil des Machine Learnings stellt selbstverständlich das Trainieren des entwickelten Modells dar. Die Schwierigkeit besteht dabei darin, dass der Lernvorgang je nach Größe, Komplexität und Umfang der Daten eine gewisse Zeitdauer erfordert. Konkret bedeutet das, dass eine Epoche bei einem Training des Modells mit knapp 2.500 Bildern mitunter 2 Stunden Zeit in Anspruch genommen hat. Infolgedessen war es schwierig und zugleich zeitaufwendig zu beurteilen, inwieweit Änderungen an dem Modell und den Parametern, die dieses beeinflussen, zu Verbesserungen oder Verschlechterungen führten.











5.7 Produzieren von brauchbaren Ergebnissen

Zum einen stellt der Ansatz des Machine Learnings durchaus einen faszinierenden Ansatz dar, da ohne konkrete Nennung von Regeln und Anweisungen, ein System geschaffen wird, das in der Lage ist, Aussagen über bekannte sowie unbekannte Daten treffen zu können. Damit einher geht allerdings auch eine gewisse Problematik, die bei Convolutional Networks und auch anderen Methodiken des Machine Learnings, die eine entsprechende Größe und Komplexität haben, auftritt. Es ist mitunter schwierig nachzuvollziehen, was sich hinter den konkreten Gewichten und Anpassungen des Systems verbirgt. Dies erschwert die Suche nach Fehlern und das Debugging immens. Auf der einen Seite stellt das Verwenden von Frameworks wie beispielsweise TensorFlow oder Scikit-Learn eine Hilfestellung dar, die dem Anwender ermöglicht, sich auf die konkreten inhaltlichen Problematiken konzentrieren zu können. Auf der an-

deren Seite bleibt dem Anwender dadurch ein großer Teil der darunterliegenden Komplexität verborgen. Dies stellt selbstverständlich je nach Betrachtungswinkel auch einen Vorteil dar. Sobald allerdings Ergebnisse entstehen, die nicht dem Erwarteten entsprechen, kann es sich als durchaus schwierig herausstellen, die Ursachen dafür aufzuspüren.

Nachfolgend werden die mit Keras erzielten Ergebnisse aufgelistet. Dabei repräsentiert die ersten Spalte den Input, die darauffolgenden drei Spalten die Ergebnisse nach der ersten, fünften und zehnten Epoche und die letzte Spalte das Originalbild, das nach Möglichkeit wiederhergestellt werden sollte.

Tabelle 1: Mit *Keras* erzielte Resultate.

Input	1. Epoche	5. Epoche	10. Epoche	Original
				
				

6 FAZIT

Die Ergebnisse, die dieses Projekt liefert, sind von einem verwendbaren Resultat noch etwas entfernt. Dennoch sieht man eine valide Approximation, die auch nach zehn Epochen noch nicht abgeschlossen zu sein scheint. Wir sind der Überzeugung, dass die Resultate, mit Anpassungen folgender Parameter, durchaus brauchbar werden können.

6.1 Trainingsdatensätze

Ideal wäre eine Datenbank ähnlich der *FaceScrub*⁸ jedoch mit mehr individuellen Personen in natürlicher Umgebung. Denn, möchte man das Netz dazu verwenden Bilder wiederherzustellen, so hat man dort keinen Einfluss auf den Kontext, in dem das Ursprungsbild aufgenommen wurde. Zudem sollte es sich als hilfreich erweisen Bilder in einer möglichst großen Auflösung zum Trainieren zu verwenden, da eine Vergrößerung der Ursprungsbilder verlustfrei möglich ist. Muss man das Ursprungsbild jedoch zuvor runterskalieren, geht an dieser Stelle schon Information verloren, die sonst dazu dienen könnte eine mögliche Ursprungsinformation wiederherzustellen.

6.2 Lerndauer

Das Netz noch mehr zu trainieren war, mit den uns zur Verfügung stehenden Computern, nicht möglich. Eine unterstützende Maßnahme hierfür könnte *Cloud Computing*²² sein, mit der es möglich wäre noch mehr Datensätze in einer noch kürzeren Zeit zu lernen.

²²Wikipedia, Cloud Computing.
(https://de.wikipedia.org/wiki/Cloud_Computing)

ABBILDUNGSVERZEICHNIS

Abbildung 1	Gleichung der gausschen Weichzeichnung zweier Dimensionen.	4
Abbildung 2	Vergleichsbild Weichzeichnen mit verschiedenen Parametern.	4
Abbildung 3	Vergleichsbild Weichzeichnen mit verschiedenen Kantenlängen.	5
Abbildung 4	Perzeptron	9
Abbildung 5	Berechnung des Outputs.	9
Abbildung 6	Mehrschichtiges neuronales Netz.	10
Abbildung 7	Berechnung des Outputs bei Verwendung eines Bias.	10
Abbildung 8	Sigmoid-Funktion.	11
Abbildung 9	Vergleich der Aktivierungsfunktionen <i>Sigmoid</i> und <i>Step-Funktion</i>	11
Abbildung 10	Hidden-Layer Darstellung.	12
Abbildung 11	Mean Squared Error (MSE).	13
Abbildung 12	Darstellung des Gradientenabstieg bei einer Funktion in Abhängigkeit von zwei Variablen.	14
Abbildung 13	Veränderung des Funktionswerts.	14
Abbildung 14	Definition des Gradienten	14
Abbildung 15	Veränderung des Funktionswerts durch Gradienten ausgedrückt	15
Abbildung 16	Feld von Inputneuronen.	16

Abbildung 17	Convolution-Prozess, Schritt 1.	17
Abbildung 18	Convolution-Prozess, Schritt 2.	17
Abbildung 19	Drei Filterschichten.	18
Abbildung 20	Maxpooling Prozess.	19
Abbildung 21	Convolution Prozess.	20
Abbildung 22	Vollständiges <i>Convolutional Network</i> mit Klassifizierungs-Ausgabe.	20
Abbildung 23	Architektur von TensorFlow	22
Abbildung 24	Exemplarischer <i>Dataflow</i>	24

TABELLENVERZEICHNIS

Tabelle 1	Mit <i>Keras</i> erzielte Resultate.	29
-----------	--	----