IMAGE DEOBFUSCATION OF GAUSSIAN BLUR AND MOSAIC

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

26. März 2018

INHALTSVERZEICHNIS

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 hevorzuheben, 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/TPU[1]/FPGA[2]) 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 weitgehends 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.

Maßgablich kommen beim Verfälschen zwei Verfahren zum Einsatz[3]: "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 verrinngert wird. Damit wird der Verlust von Detailinformationen erreicht. Die mathematische Formel, nach der die Transformation funktioniert, lautet:

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

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 Abbildung1). 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.

Original $\sigma = 3$ $\sigma = 6$

Abbildung 1: 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 Verfahrensfa-

milie 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. 2

Abbildung 2: Vergleichsbild Weichzeichnen mit verschiedenen Kantenlängen.

Der in diesem Verfahren betrachtete Parameter (siehe Abbildung2) 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 augenommen worden sind, und hält neben Metadaten über Pose, Geschlecht, Ethnie und Alter auch weiterführende Daten bereit wie Augenposition und Kamerawinkel. Die Daten liegen im Portable Pixmap Format RGB- und im Graustufenformat

[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

homogen in einer vor Maximalauflösung von *512x768* Pixeln vor.

Die *FaceScrub*[8] umfasst über 100.000 Gesichtsbilder von 530 prominenten Menschen, die in einer unkontrollierten Umgebung augenommen worden sind und stellt keine weiteren Metadaten bereit. Die Daten liegen im JPEG-Format in einer vor Maximalauflösung von *512x768* Pixeln vor.

Um trotz der vergleichsweise geringen Datenmenge. Vergleichbare Projekte [9,10] 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 großten Teildatensatz ausmachen.

Die benötigten Testdatensätze wurde mithilfe von ImageMagick[11] 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 Trainigsdaten noch eine Auflösung von 64x96 Pixeln haben. Es wurden vier unterschiedliche Testdatensätze mit folgenden CLI-Befehlen generiert [12]:

[9] Richard McPherson, Rezar Shokri, Vitali Shmatikov, Defeating Image Obfuscation with Deep Learning. (https://arxiv.org/pdf/1609.00408.pdf)

[10] "Jenkspt", Enhancer. (https://github.com/jenkspt/enhancer)

[11] ImageMagick Studio LLC, ImageMagick. (https://www.imagemagick.org/)

[12] Das folgende BASH-Skript scripts/create_images.sh erzeugt die Testdaten. Notwendig hierfür sind die Pakete ïmagemagickünd ïmagemagick-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 \
    -separate \
    -average \
    -scale 12.5\% \
    -gaussian-blur 0x3 \
    <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>
```

CNNs: https://adeshpande3.github.io/A-Beginnerhttp://cs231n.github.io/convolu
networks/ https://ujjwalkarn.me/2016/08/11/intuitive-explanation-
convnets/ http://deeplearning.net/tutorial/lenet.html https://deeplearning4j.org/con

NNs: http://neuralnetworksanddeeplearning.com/chap1.html
http://neuralnetworksanddeeplearning.com/ http://www.deeplearningbook.org/

3 NEURONALE NETZE

Im Rahmen des maschinellen Lernens stellen die sogenannten 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 sogenannte *Perzeptron* - dieses stellt die kleinste Einheit eines neuronalen Netztes dar und wird auch als "künstliches Neuron" bezeichnet.

Grundsätzlich akzeptiert ein Neuron einen beliebig großen Input bestehend aus Features *x1*, *x2*, ..., *xn* und berechnet daraus ein Ergebnis.

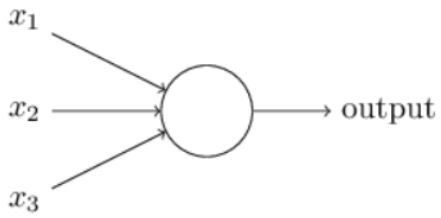


Abbildung 3: Perzeptron

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:

$$\text{output} = \begin{cases} 0 & \text{if } \sum_j w_j x_j \leq \text{ threshold} \\ 1 & \text{if } \sum_j w_j x_j > \text{ threshold} \end{cases}$$

Abbildung 4: Neuron mit drei Inputgrößen.

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 Model komplexer zu gestalten, indem meh-

rere Perzeptrons miteinander verknüpft werden, so dass beispielsweise das nachfolgende Netzwerk entstehen könnte:

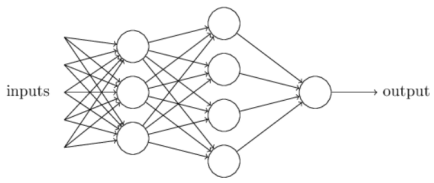


Abbildung 5: Mehrschichtiges neuronales Netz.

In Grafik[4] wurde ein Schwellwert eingeführt, der überschritten werden muss, damit ein Perzeptron aktiviert wird. Um das Modell zu vereinheitlichen, kann der *Bias* definiert werden, der den negativen Schwellwert darstellt. Durch diese Maßnahme kann die Aktivierungsfunktion des Perzeptrons dann geschrieben werden als:

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

Abbildung 6: Berechnung Schwellwertfunktion.

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

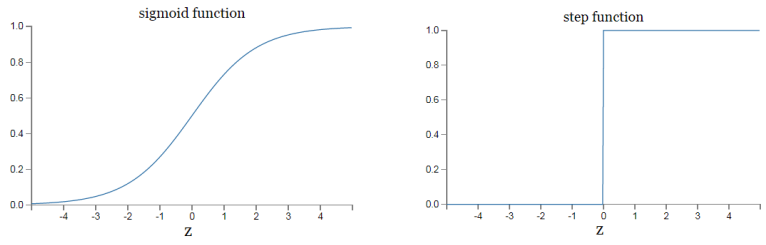
zur Folge haben, sondern folgenlos bleiben können bis irgendwann ein Sprung von 0 auf 1 oder umgekehrt stattfindet, was wiederum eine große Änderung darstellt.

Die Weiterentwicklung besteht nun in einer Verfeinerung der Aktivierungsfunktion. Anstatt eine Sprungfunktion[8b] zu verwenden, die lediglich 0 und 1 als Funktionswert annehmen kann, wird die sogenannte Sigmoid Funktion[8a] eingeführt, die die folgende Form hat:

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

Abbildung 7: Sigmoid-Funktion.

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



Sigmoid Funktion. Step Funktion.

Abbildung 8: 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 28x28 Pixel klassifiziert werden und es gibt 10 mögliche Klassen, dann besteht das Input-Layer aus 28x28 = 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:

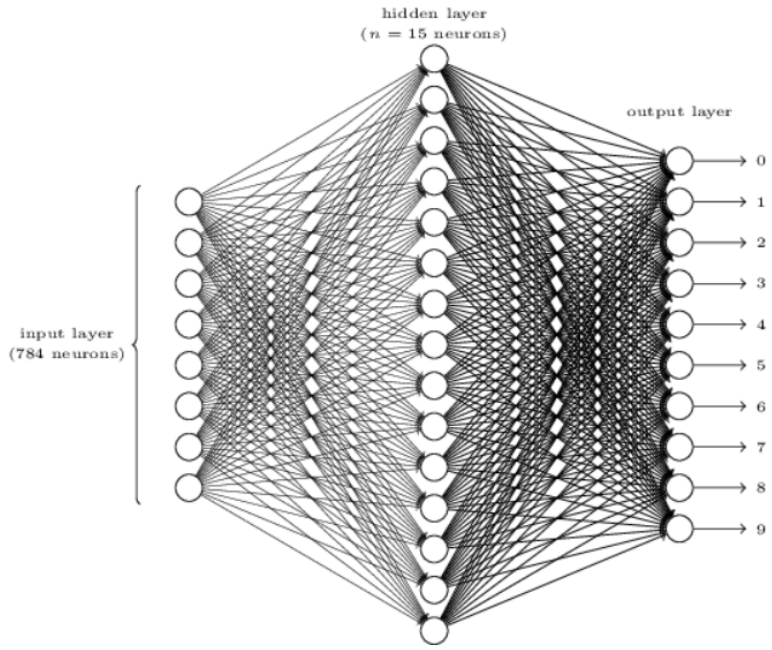


Abbildung 9: Hidden-Layer Darstellung.

3.4 Lernen – Das Anpassen der Gewichte

Das Lernen stellt den zentralen Ansatz von neuronalen Netzes 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 Verwendeung 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 10: Lern-Funktion.

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 Richtung $v1$ und etwas in Richtung $v2$ so gilt für den Funktionswert C:

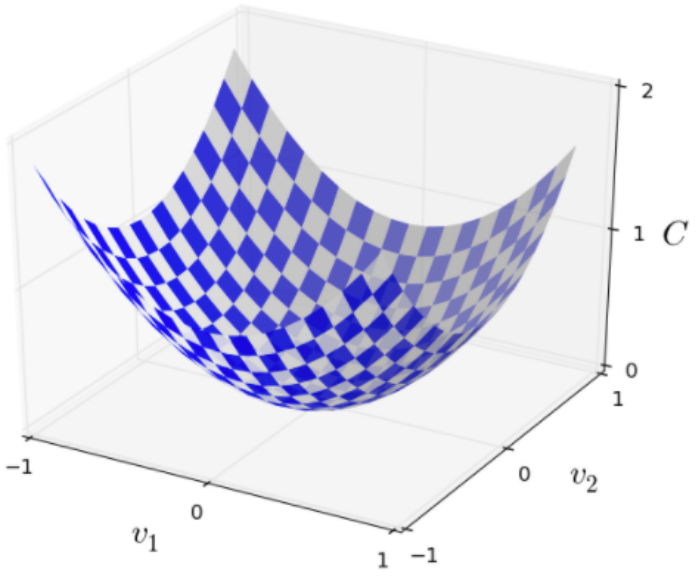


Abbildung 11: 3D Darstellung des Gradientenabstiegs.

Der Gradient wird dann folgendermaßen definiert:

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

Abbildung 12: Gradientendefinition.

Durch die Verwendung dieses mathematischen Objekts kann die vorherige Gleichung[12] formuliert werden als:

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

Abbildung 13: Perzeptron

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.

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 28x28 Pixel betrachtet entspräche das dem nachfolgenden Input-Layer:

Im Unterschied zu *herkömmlichen* neuronalen Netzen, bei denen üblicherweise alle Input-Neuronen mit allen Neuronen des

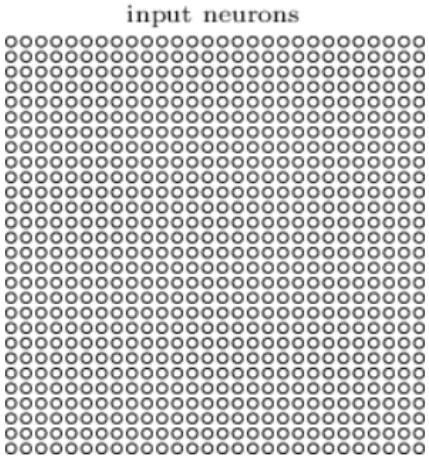


Abbildung 14: Feld von Inputneuronen.

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:

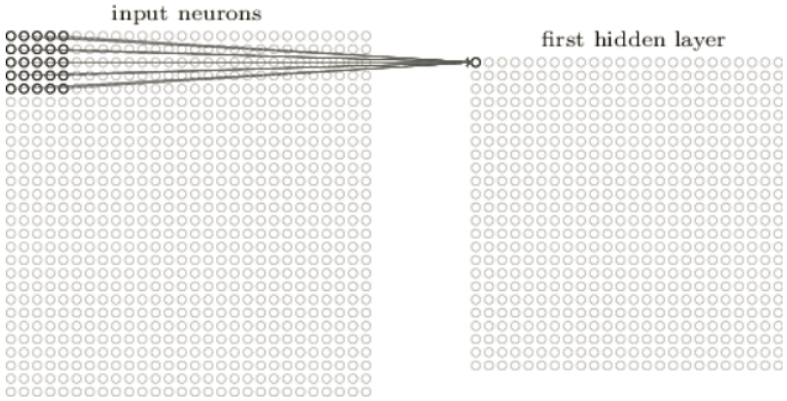


Abbildung 15: Convolution-Prozess, Schritt 1.

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 28x28 Pixel und einem lokalen rezeptiven Feld der Größe 5x5

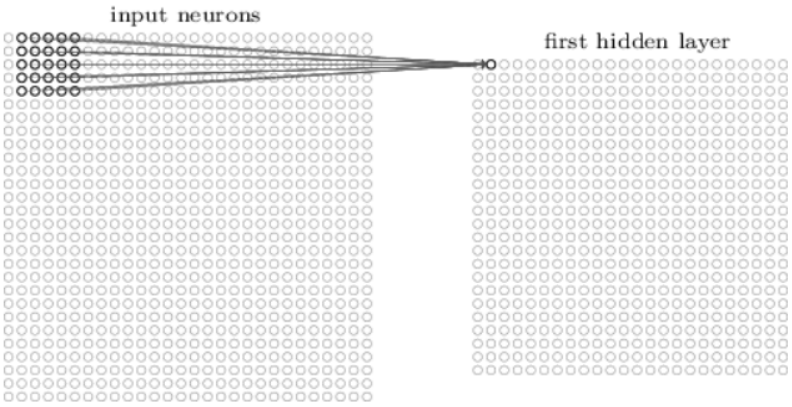


Abbildung 16: Convolution-Prozess, Schritt 2.

Pixel aus, so folgt daraus ein erstes Hidden-Layer der Größe 24x24 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 5x5 Pixel - über 5x5 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 (TODO: Begriff einführen)).

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

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

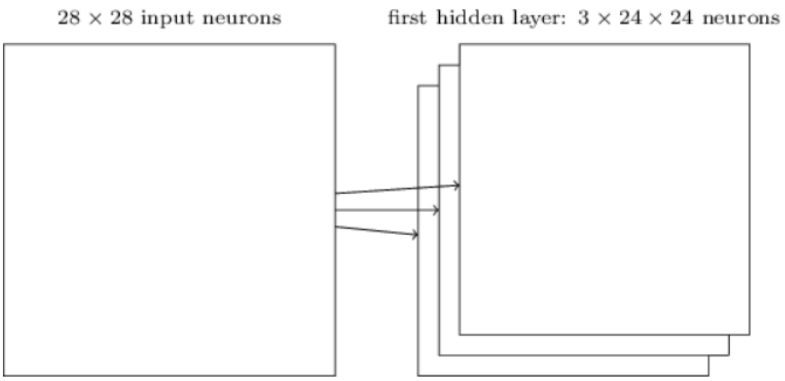


Abbildung 17: Drei Filterschichten.

schnelleres Lernen ermöglicht werden kann. Eine einzige feature map (TODO) würde bei den bisher betrachteten Dimensionen durch 5x5 = 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* 20x26 = 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 x 28) x 30 + 30 = 23550 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 enstehen, werden gemeinhin als *convolutional* Layer bezeichnet. Von diesen kann ein weiterer wesentlicher Bestandteil von convolutional networks abgegrenzt werden - die sogenannten *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, dass sich über das Ergebnis des vorherigen Schrittes bewegt. Es könnte sich beispielsweise um ein Feld der Größe 2x2 handeln. Dieses würde somit immer 4 Neuronen betrachten und - im Falle des sogenannten max-

poolings (einer konkreten Ausprägung des Poolings) - das größte von ihnen auswählen.

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

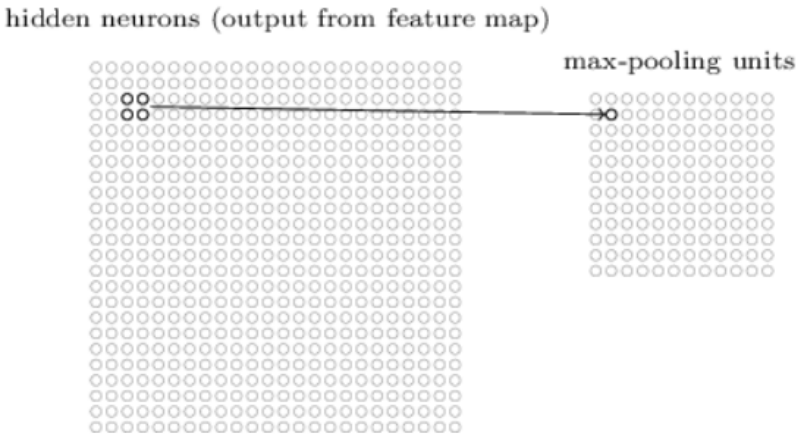


Abbildung 18: Maxpooling Prozess.

Dadurch, dass vier Neuronen auf ein Neuron projeziert werden, verringert sich die Größe bei diesem konkreten Beispiel von 24x24 auf 12x12. Dieses Vorgehen wird selbstverständlich auf alle feature maps des vorherigen convolutional Layers angewendet, so dass das folgende Ergebnis entsteht.

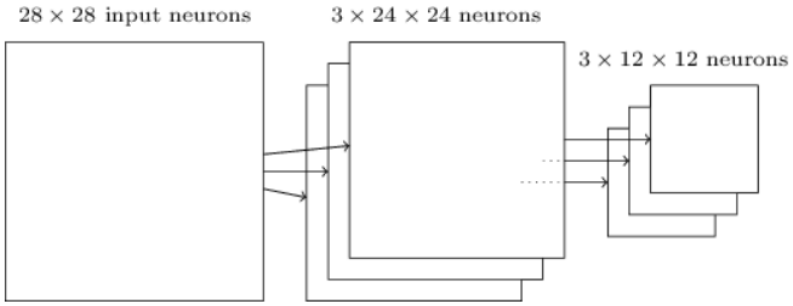


Abbildung 19: Convolution Prozess.

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

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:

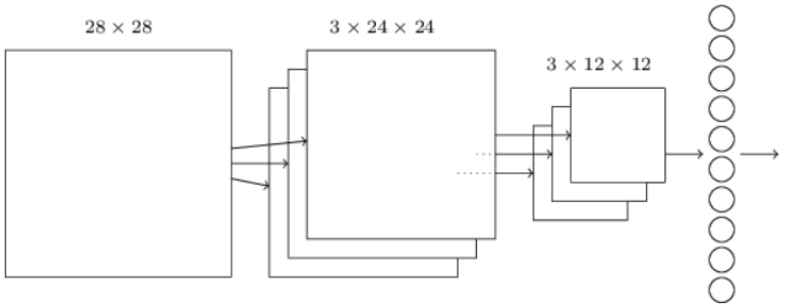


Abbildung 20: Vollstäniges *Convolutional Network mit Klassifizierungs-Ausgabe.*

ABBILDUNGSVERZEICHNIS