Emotionserkennung mit künstlichen neuronalen Netzen

wissenschaftlich-praktische Arbeit

Gustav Lahmann

Georg Cantor Gymnasium Halle (Saale)

8. Juni 2018

Betreut durch: Herr Dipl.-Ing. (FH) Dirk Hesselbach *(Hochschule Merseburg)*

Inhaltsverzeichnis

1 Einleitung

Während der Arbeit in einer AG mit dem Thema künstliche Blumen bauten und programmierten die Mitglieder einen Kaktusroboter. Dabei entstand die Idee, dass er auf die Emotionen der Menschen in seiner Umgebung reagieren soll. Dieses Vorhaben wurde als Thema der wissenschaftlich Praktischen Arbeit aufgegriffen.

1.1 Zielsetzung

Das Ziel dieser Arbeit ist die Emotionserkennung in Gesichtern aus Bildern mit Hilfe von künstlichen neuronalen Netzen (KNN). Im aufgenommenen Kamerabild sollen Gesichter gefunden und anschließend jeder der Emotionen *fröhlich*, *traurig*, *überrascht*, *zornig*, *angewidert*, *verachtend* und *ängstlich* ein Score zugeordnet werden, der angibt, mit welchem Anteil die jeweilige Emotion im Gesicht erkannt wurde. Der größte Score einspricht dabei immer der Emotion, die im Bild hauptsächlich vorliegt. Diese Emotionen entsprechen den Basisemotionen nach Paul Ekman und werden kulturübergreifend verstanden.[1] Da das KNN mit einer festen Zuordnung einer der oben genannten Emotionen zu jedem Beispielbild trainiert wird, muss diese Hauptemotion nicht immer der tatsächlichen entsprechen, da diese eine Kombination aus mehreren sein kann.

1.2 Verwendung

Unternehmen wie Affectiva nutzen die Emotionen des Nutzers um die Wirkung von Werbespots und Computerspielen zu beurteilen oder Stresssituationen beim Autofahren zu erkennen.[2] Weitere Anwendungen sind die Windows Fotogalerie oder Snapchat Filter. In der Auswertung von Daten aus Überwachungssystemen zur Erkennung von Konflikten oder in Tests zur Benutzerfreundlichkeit von Software ist eine Verwendung ebenfalls denkbar.

2 Vom Problem zum Programm

Ziel ist es, ein alleinstehendes Modul zur Emotionserkennung zu schaffen. Es kann später in den Roboter intrigiert werden und die Ergebnisse an andere Mikrorechner im System weiterleiten.

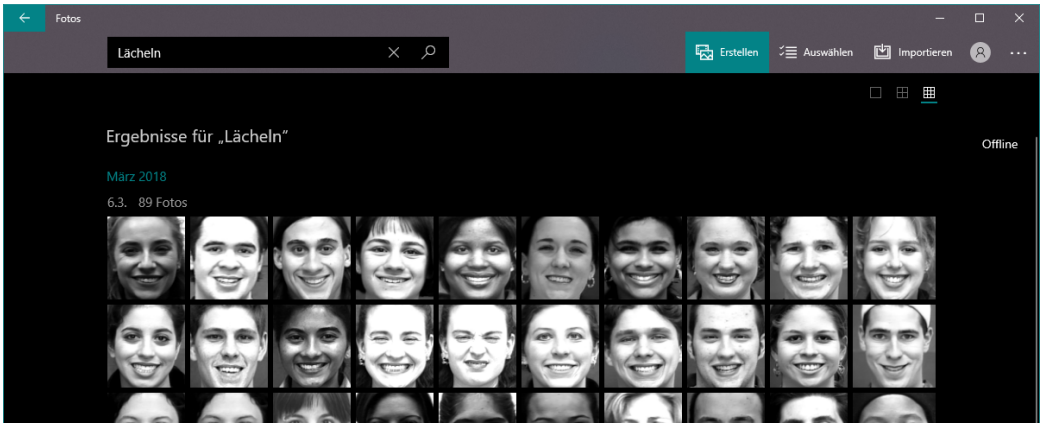
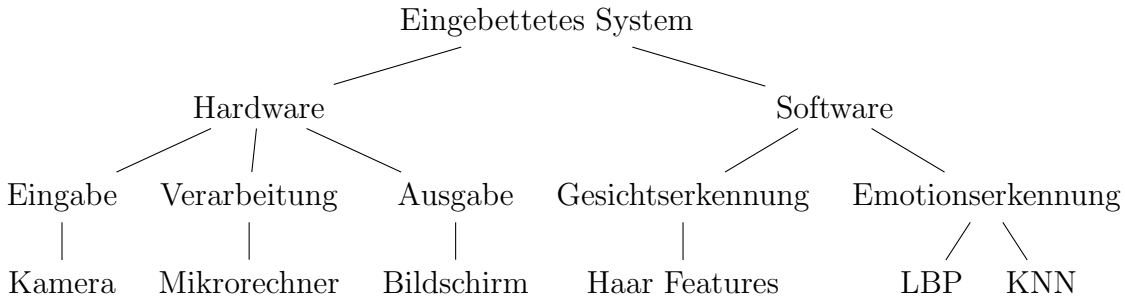


Abbildung 1: Die Windows 10 Fotogalerie erkennt lächelnde Gesichter aus meinen Test- und Trainingsdaten und fasst sie in einem Album zusammen

2.1 Problemanalyse

Das eingebettete System besteht aus einem Hardware und Software Teil. Als Erstes wird über eine Kamera ein Bild aufgenommen und an die Software übertragen. Diese wertet das Bild aus und erkennt zunächst mit Hilfe von Haar Features Gesichter im Bild. Anschließend wird das statistisches Auswertungsverfahren "Local Binary Pattern"(LBP) angewendet und die erhaltenen Häufigkeiten als Input für das KNN verwendet. Es wurde mit vorher klassifizierten Testbildern trainiert und erkennt die Emotion. Das Ergebnis wird anschließend grafisch als Balkendiagramm auf einem Display ausgegeben.



2.2 Technische Basis

Das möglichst minimal gehaltene System, welches verwendet wurde, besteht aus einem *Raspberry Pi 3 Model B* (64bit QuadCore CPU mit 1,2 GHz; 1 GB RAM) mit dem *Raspberry Pi Camera Module v2* (Auflösung: 8 MP) und einem *SmartiPi Touch Display*. Von Vorteil ist neben dem Preis und der Größe

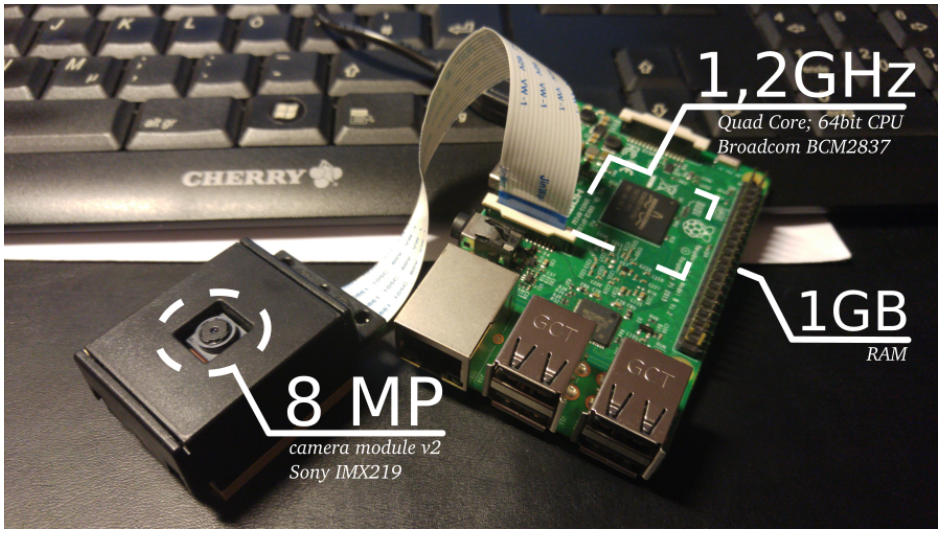


Abbildung 2: Raspberry Pi mit angeschlossenem Kameramodul

die Möglichkeit eine Python Bibliothek zur Steuerung der Kamera verwenden zu können. Diese lässt sich gut in die verwendeten Grafikbibliothek OpenCV integrieren. Das Betriebssystem Raspian stellt alle weiteren erforderlichen Bibliotheken für Python außer OpenCV zur Verfügung, welche man selbst aus dem Quellcode kompilieren muss.

2.3 Vorbereitung

Bevor eine Klassifizierung durch das KNN stattfinden kann, müssen aus dem aufgenommenen Bild alle Gesichter extrahiert und auf einen gemeinsamen Standard normalisiert werden.

2.3.1 Aufnahme des Kamerabildes

Die Kamera liefert einen kontinuierlichen Datenstrom von Farbbildern mit einer Auflösung von bis zu 3280×2464 Pixeln.[3] Dieses Bild wird als Pixelmatrix gespeichert und weiter verarbeitet. Dabei wird das nächste Bild erst erfasst, wenn alle Gesichter im aktuellen Bild erkannt und ausgewertet wurden.

2.3.2 Bildverarbeitung

OpenCV speichert Bilder als zweidimensionale Matrizen. Die Anzahl der Spalten und Zeilen entsprechen dabei der Breite und Höhe des Bildes. Bei

Graustufenbildern werden in der Matrix die Grauwerte gespeichert. In Farbbilder mit RGB-Farbsystem wird zu jedem Pixel ein Tripel mit Blau-, Grün- und Rotwert gespeichert.

$$A = \begin{pmatrix} (B,G,R)_{00} & (B,G,R)_{01} & (B,G,R)_{02} \\ (B,G,R)_{10} & (B,G,R)_{11} & (B,G,R)_{12} \\ (B,G,R)_{20} & (B,G,R)_{21} & (B,G,R)_{22} \end{pmatrix} \text{ Bildmatrix der Größe } 3 \times 3$$

Die folgenden Verfahren (2.3.4 und 2.4) verwenden *Faltungsmatrizen*. Dabei handelt es sich um Matrizen mit ungerader Kantenlänge, welche an jeder Position der Bildmatrix angewendet werden. Dazu wird die Faltungsmatrix pixelweise auf der Bildmatrix verschoben und für den jeweiligen Pixel werden Rechenoperationen durchgeführt.

2.3.3 Graustufenumwandlung

Für den Gesichtsausdruck ist nur die Struktur des Gesichts und nicht die Farbe wichtig. Aus diesem Grund wird die Datenmenge des Bildes auf die Grauwertinformationen reduziert. Dazu wird die Funktion cv2.cvtColor() [4] mit dem Parameter cv2.COLOR_BGR2GRAY verwendet. Sie berechnet einen Grauwert Y zu jedem Farbwert (R, G, B):

$$Y = 0,299 \cdot R + 0.587 \cdot G + 0.144 \cdot B$$

Die Gewichtung der verschiedenen Farbkanäle entspricht der Empfindung des menschlichen Auges.[5]

2.3.4 Gesichtserkennung mit Haar Kaskaden

Ein effizienter Algorithmus zur Gesichtserkennung in einem Bild mit Hilfe von Haar Features wurde von Paul Viola und Michael Jones 2001 veröffentlicht.[6] Dabei werden über das Grauwertbild Faltungsmatrizen verschoben. Die Faltungsmatrizen entprechen Haar Features und bestehen aus schwarzen, sowie weißen Bereichen. An jeder Position werden alle Grauwerte im schwarzen Bereich aufsummiert und anschließend von der Summer der Grauwerte im weißen Bereich abgezogen. Daraus entsteht eine Differenz, welche angibt, wie stark das jeweilige Feature ausgeprägt ist. Sind die beiden Bereiche sehr ähnlich, liegen die Summen in der gleichen Größenordnung und die Differenz ist folglich klein. Wenn die Bereiche dem Muster des Features folgen, unterscheiden sich die Summen stark voneinander und damit wird der Betrag der Differenz groß. Um möglichst alle Größen der Merkmale abzudecken, die

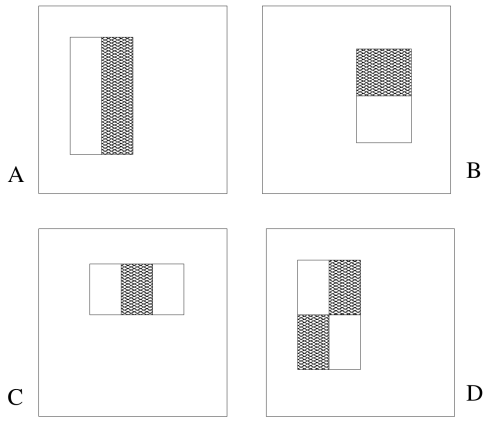


Abbildung 3: Haar Features[6]

ein Feature beschreibt, werden sie für den gesamten Prozess auf verschiedene Größen skaliert.

Die so errechneten Mengen der verschiedenen Features werden mit dem Machine Learning Verfahren „AdaBoost“ in relevante und unrelevante Features zum Erkennen von Gesichtern sortiert. Kombiniert bilden viele schwache Klassifizierer dann einen starken, welcher verwendet wird, um neue Gesichter zu erkennen. Das Berechnen der vielen Features für jedes Gesicht ist rechenaufwändig. Deshalb werden Regionen des Bildes verworfen auf welche die wichtigsten Features nicht zutreffen, um Bereiche die das Gesicht eher enthalten gründlicher Untersuchen zu können.

2.3.5 Skalierung

Um unterschiedliche Entfernungen der Gesichter von der Kamera zu kompensieren, werden die erkannten Gesichter auf eine festgelegte Größe von 100×100 Pixel skaliert. Diese Bildgröße beinhaltet ausreichend viele Details und ermöglicht eine schnelle Verarbeitung. Sie wurde experimentell bestimmt und ist ein Kompromiss. Dabei wurden die notwendige Pixelanzahl für außreichend Details und geringe Speichernutzung, die Blockgröße des LBP und die Neuronenanzahl des KNN berücksichtigt.

2.4 Local Binary Pattern

Das *Local Binary Pattern* Verfahren ist eine statistische Auswertung des Bildes welches die Datenmenge stark reduziert. Dabei behält es die Kanten als wichtigste Information der Gesichtszüge bei.

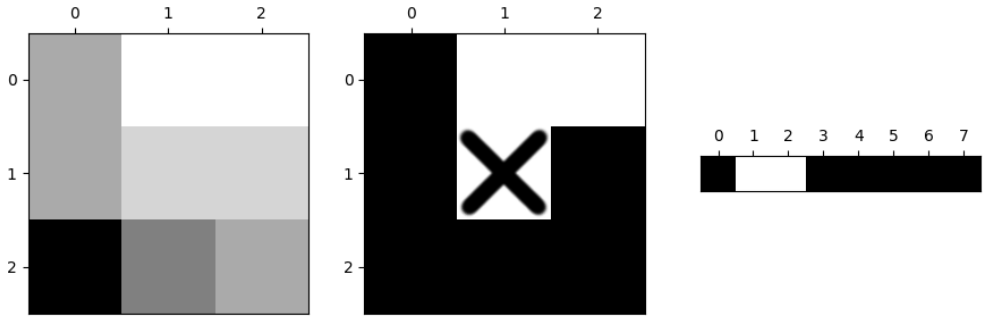


Abbildung 4: 3×3 Ausschnitt aus dem Grauwertbild, nach dem Vergleichen und als Pattern $(01100000)_2$ in grafischer Darstellung

Der folgende Algorithmus nutzt einen 3×3 Ausschnitt, welcher an jede Position des Bildes verschoben wird. Dort werden jeweils alle äußeren Pixel mit dem mittleren verglichen. Ist der Grauwert des äußeren Pixels größer (heller), wird dieser auf 1 (weiß), ansonsten auf 0 (schwarz) gesetzt. Das zugehörige Pattern entsteht aus diesen Werten beginnend mit der linken oberen Ecke im Uhrzeigersinn gelesen. In Abbildung 4 ist eine Faltungmatrix dargestellt, deren Local Binary Pattern $(01100000)_2$ entspricht.

Local Binary Pattern

coordinates {Liste der Koordinaten als Tupel} *matrix* {3×3 Matrix der Grauwerte des Kernels} *pattern* {Liste der Binären Vergleichsergebnisse}	
$coordinates \leftarrow (0,0),(0,1),(0,2),(1,2),(2,2),(2,1),(2,0),(1,0)$	
$\forall coordinate \in coordinates$	
$matrix[coordinate] > matrix[(1,1)]$	
WAHR	FALSCH
1 zu *pattern* hinzufügen	0 zu *pattern* hinzufügen

2.4.1 Uniforme Pattern

Als uniforme Pattern werden alle Pattern bezeichnet, die höchstens zwei Übergänge von 0 zu 1 oder 1 zu 0 enthalten.[7]

Deshalb ist zum Beispiel $(00000101)_2 = (5)_{10}$ kein uniformes Pattern, $(01100000)_2 = (96)_{10}$ aber schon. Bei einer 8-bit Zahl ergeben sich 58 uniforme Pattern, welche in der folgenden Tabelle mit ihrer Binärdarstellung aufgeführt sind.

$(0)_{10}$	$(00000000)_2$	$(62)_{10}$	$(00111110)_2$	$(207)_{10}$	$(11001111)_2$
$(1)_{10}$	$(00000001)_2$	$(63)_{10}$	$(00111111)_2$	$(223)_{10}$	$(11011111)_2$
$(2)_{10}$	$(00000010)_2$	$(64)_{10}$	$(01000000)_2$	$(224)_{10}$	$(11100000)_2$
$(3)_{10}$	$(00000011)_2$	$(96)_{10}$	$(01100000)_2$	$(225)_{10}$	$(11100001)_2$
$(4)_{10}$	$(00000100)_2$	$(112)_{10}$	$(01110000)_2$	$(227)_{10}$	$(11100011)_2$
$(6)_{10}$	$(00000110)_2$	$(120)_{10}$	$(01111000)_2$	$(231)_{10}$	$(11100111)_2$
$(7)_{10}$	$(00000111)_2$	$(124)_{10}$	$(01111100)_2$	$(239)_{10}$	$(11101111)_2$
$(8)_{10}$	$(00001000)_2$	$(126)_{10}$	$(01111110)_2$	$(240)_{10}$	$(11110000)_2$
$(12)_{10}$	$(00001100)_2$	$(127)_{10}$	$(01111111)_2$	$(241)_{10}$	$(11110001)_2$
$(14)_{10}$	$(00001110)_2$	$(128)_{10}$	$(10000000)_2$	$(243)_{10}$	$(11110011)_2$
$(15)_{10}$	$(00001111)_2$	$(129)_{10}$	$(10000001)_2$	$(247)_{10}$	$(11110111)_2$
$(16)_{10}$	$(00010000)_2$	$(131)_{10}$	$(10000011)_2$	$(248)_{10}$	$(11111000)_2$
$(24)_{10}$	$(00011000)_2$	$(135)_{10}$	$(10000111)_2$	$(249)_{10}$	$(11111001)_2$
$(28)_{10}$	$(00011100)_2$	$(143)_{10}$	$(10001111)_2$	$(251)_{10}$	$(11111011)_2$
$(30)_{10}$	$(00011110)_2$	$(159)_{10}$	$(10011111)_2$	$(252)_{10}$	$(11111100)_2$
$(31)_{10}$	$(00011111)_2$	$(191)_{10}$	$(10111111)_2$	$(253)_{10}$	$(11111101)_2$
$(32)_{10}$	$(00100000)_2$	$(192)_{10}$	$(11000000)_2$	$(254)_{10}$	$(11111110)_2$
$(48)_{10}$	$(00110000)_2$	$(193)_{10}$	$(11000001)_2$	$(255)_{10}$	$(11111111)_2$
$(56)_{10}$	$(00111000)_2$	$(195)_{10}$	$(11000011)_2$		
$(60)_{10}$	$(00111100)_2$	$(199)_{10}$	$(11000111)_2$		

Diese Pattern stellen Kanten dar, während alle nicht uniformen Pattern eher weiche Übergänge beschreiben.

2.4.2 Patternhäufigkeit

Um eine das Gesicht repräsentierende Statistik zu erhalten, wird das Bild in quadratische Blöcke mit einer festgelegten Seitenlänge unterteilt. Bei einer Blockgröße von beispielsweise 25 Pixeln und einer Bildgröße von 100×100 Pixel ergibt das $\frac{100}{25} \cdot \frac{100}{25} = 4 \cdot 4 = 16$ Blöcke. Für jeden dieser Blöcke wird gezählt, wie viele Pattern jeder Art darin vorkommen. Diese Häufigkeiten werden in einer Liste gespeichert, wobei Index 0 bis 57 den jeweiligen Pattern zugeordnet und in Index 58 alle nicht uniformen Pattern gezählt werden.

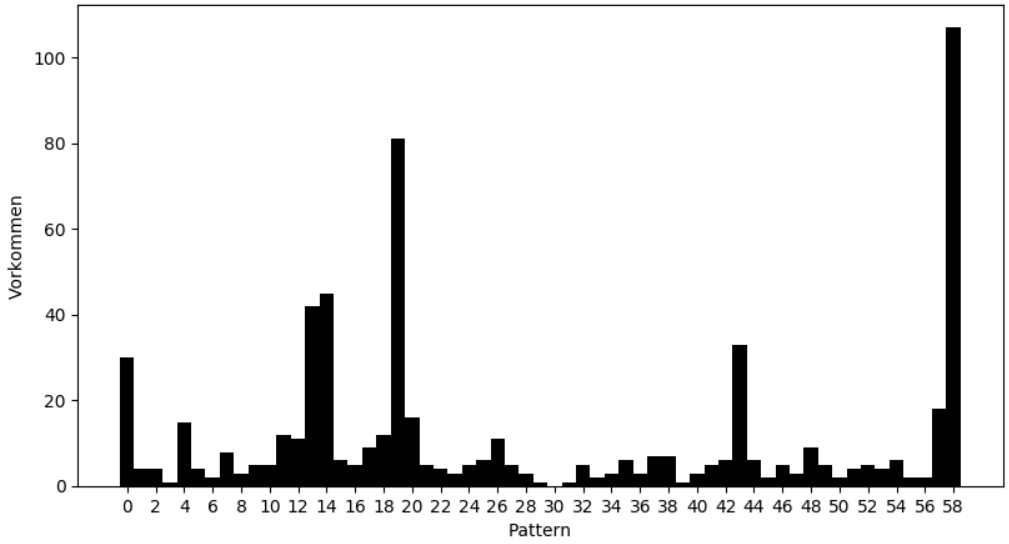


Abbildung 5: Absolute Häufigkeitsverteilung der Pattern im ersten Block (umrahmt) als Histogramm

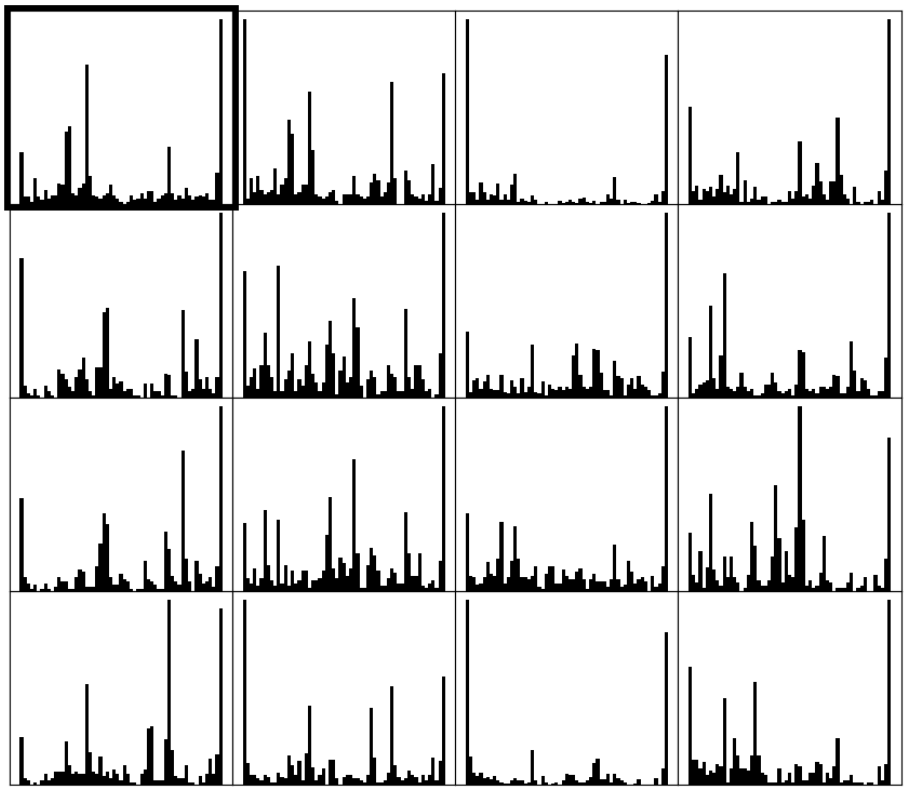


Abbildung 6: Histogramme für alle 16 Blöcke des Bildes

Durch dieses Vorgehen wird eine Häufigkeitsverteilung erstellt, die verschiedene Kantenarten erfasst und andere (nicht uniforme) Übergänge als ein Kriterium vereinfacht und somit die Anzahl der Klassen von 256 auf 59 reduziert. Während die Häufigkeiten die Merkmale des Blocks beschreiben, ist der Block an sich für die Speicherung des Ortes wo dieses Merkmal auftritt zuständig. Deshalb werden im letzten Schritt alle Häufigkeiten der Blöcke in einer festgelegten Reihenfolge zum Eingabevektor für das künstliche Neuronale Netz in 2.5 zusammengefügt.

$$X = \begin{pmatrix} H_{Block0,Pattern0} \\ \vdots \\ H_{Block0,Pattern58} \\ \vdots \\ H_{Block15,Pattern0} \\ \vdots \\ H_{Block15,Pattern58} \end{pmatrix}$$

2.5 Auswertung

Aus der statistischen Repräsentation des Gesichts soll jetzt mit Hilfe eines KNNs eine Relation zu der jeweilig vorliegenden Emotion hergestellt werden. Dabei wird dem Eingabevektor aus 2.4.2 ein 7-dimensionaler Zielvektor Y zugeordnet, der überall außer bei der richtigen Emotion, wo der Wert 1 steht, eine 0 enthält. Mit überwachtem Lernen wird das Netz auf die Verbindung zwischen den beiden trainiert.

2.5.1 FeedForward KNNs

FeedForward Netze sind künstliche neuronale Netze und bestehen aus einer Eingabeschicht, mehreren versteckten Verarbeitungsschichten und einer Ausgabeschicht. Die jeweilige Schicht besteht aus mehreren Neuronen. Jedes Neuron ist mit allen anderen Neuron der folgenden Schicht über Kanten verbunden. Zwischen einer Schicht mit 4 Neuronen und einer mit 3 Neuronen existieren also $4 \cdot 3 = 12$ Kanten.

Um ein Ergebnis aus dem Netz zu erhalten, ist eine Eingabe nötig. Diese ist der vorher auf Werte von 0 bis 1 normalisierte Eingabevektor X. Die im Vektor enthaltenen Zahlen entsprechen dabei den Werten der Neuronen der Eingabeschicht.

2.5.2 Gewichte

Die Gewichte repräsentieren die Stärke der Verbindung (Kanten) zwischen zwei Neuronen i und j und speichern das gelernte Wissen. Diese Verbindung wirkt somit verstärkend oder hemmend auf die Aktivität des Neurons j.

Die Eingaben sind die Produkte aus den Aktivierungen $a_i, ..., a_{i_n}$ aller vorherigen Neuronen $i_1, ..., i_n$ und den Gewichten $w_{ji_1}, ..., w_{ji_n}$ die diese Neuronen mit j verbinden, zur Netzeingabe z_j aufsummiert. [8, S. 37]

$$z_j = \sum_{i=1}^{n} w_{ji} \cdot a_i$$

Aus den Aktivierungen der Neuronen einer Schicht als n-dimensionaler Vektor A und einer Gewichtsmatrix W mit der Größe $m \times n$ berechnet sich der m-dimensionale Eingabevektor Z der nächsten Schicht.

$$Z = W \cdot A$$

$$\begin{pmatrix} z_1 \\ \vdots \\ z_m \end{pmatrix} = \begin{pmatrix} w_{11} & \dots & w_{1n} \\ \vdots & \ddots & \vdots \\ w_{m1} & \dots & w_{mn} \end{pmatrix} \cdot \begin{pmatrix} a_1 \\ \vdots \\ a_n \end{pmatrix} = \begin{pmatrix} w_{11} \cdot a_1 + \dots + w_{1n} \cdot a_n \\ \vdots \\ w_{m1} \cdot a_1 + \dots + w_{mn} \cdot a_n \end{pmatrix}$$

Die Python Bibliothek *numpy* ermöglicht es für die Matrixmultiplikation mehrere Kerne parallel zu nutzen. Da die Mikroprozessoren für Matrixberechnungen optimiert sind, werden die Formeln in Matrixdarstellungen umgewandelt.

2.5.3 Bias

Die Aktivierung eines Neurons ist abhängig von den Aktivitäten der Neuronen in der vorherigen Schicht. Um zu modellieren, dass manche Neuronen generell verschiedene Aktivitätsgrade besitzen, fügt man einen Schwellwert ein, der steuert ab wann ein Neuron zu feuern anfängt. Es besteht die Möglichkeit diesen Schwellwert von der Netzeingabe z_j abzuziehen. Damit der Schwellwert mit dem gleichen Verfahren, wie die Gewichte beim Training anpassen werden kann, wird er als Gewicht eines unabhängigen, zusätzlichen Neurons in der vorherigen Schicht implementiert. Dieses Biasneuron hat immer die Ausgabe $a_{bias} = 1$. Der Schwellwert ist also nur vom Wert des Gewichts vom Biasneuron zum jeweiligen Neuron abhängig. Da alle Neuronen einer Schicht mit allen Neuronen der vorherigen Schicht verbunden sind, reicht ein Biasneuron pro Schicht um für die gesamte nächste Schicht Schwellwerte bereitzustellen.

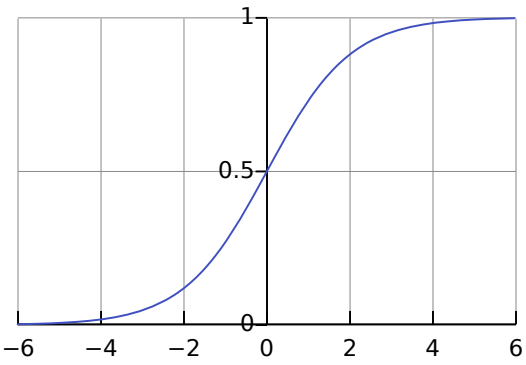


Abbildung 7: Sigmoid Funktion[9]

2.5.4 Aktivierungsfunktion

Die Aktivierungsfunktion ϕ berechnet aus der Netzeingabe z die Aktivierung a.

$$a = \phi(z)$$

Natürliche Neuronen feuern nicht über einen bestimmten Maximalwert hinaus. Deshalb bildet die Aktivierungsfunktion Zahlenwerte in einem festen Bereich ab.

Häufig wird die Sigmoid Funktion verwendet, da sie die Eingabe auf Werte von 0 bis 1 normalisiert. Sie ist im Bereich um 0 sehr sensibel und normalisiert stark bei Betragsmäßig großen Eingaben. Eine weitere Bedingung an die Aktivierungsfunktion ist die Differenzierbarkeit, welche für die Gewichtsanpassung in 2.5.5 nötig ist.

$$\phi(z) = sigmoid(z) = \frac{1}{1 + e^{-z}}$$

2.5.5 Training

Beim Trainieren eines künstlichen neuronalen Netzes werden nacheinander Bilder aus einem Trainingsset zu denen das gewünschte Ergebnis bekannt ist ins Netz eingegeben. Das gleiche Trainingsset kann in mehreren Durchläufen (Epochen) wiederholt werden, um im Gradientensbstiegsverfahren den Fehler des Netzes zu minimieren. Dabei muss die Reihenfolge immer zufällig sein, um das Lernen der Abfolge von Eingaben zu vermeiden.

Nach dem Training wird mit Testdaten überprüft, wie gut das Netz abstahiert hat, indem man dem Netz noch unbekannte Gesichter verwendet. Dabei werden die Gewichte nicht angepasst, sondern nur das Resultat des

Netzes ausgewertet. Als Testscore speichert man, wie viele der Trainingsdaten richtig klassifiziert wurden, wobei das Neuron mit der größten Ausgabe der vom Netz erkannten Emotion entspricht. Wenn der Fehler beim Training zwar gering ist, der Score beim Testen aber schlecht, wurde das Netz mit zu vielen Epochen übertrainiert und kann deshalb zwar die bekannten Gesichter erkennen, verfügt aber nicht über allgemeines Wissen, welches für unbekannte Gesichter nötig ist.

Fehler Um nach Eingabe eines Trainingsbeispiels in das KNN die Gewichte so anpassen zu können, dass sie das Resultat des Netzes verbessern, muss der Fehler des jeweiligen Beispiels gemessen werden. Er ergibt sich als Differenz aus dem Zielvektor Y und dem Ausgabevektor A.

$$E = Y - A$$

Backpropagation Bevor die Gewichte angepasst werden, muss der Fehler für jedes Neuron berechnet werden. Dies ist bis jetzt nur in der Ausgabeschicht geschehen, wo eine direkte Berechnung möglich ist. Die Backpropagation führt die Fehler der Ausgabeschicht in die vorherigen Schichten zurück. Dabei wird der Fehler eines Neurons j so auf die mit ihm verbundenen Neuronen $i_1, \ldots, i_n$ zurückgeführt, dass die durch stärkere Gewichte verbundenen, und damit mehr am Fehler beteiligten, Neuronen einen größeren Anteil am Fehler erhalten. Wenn der Fehler mit dem verbindenden Gewicht multipliziert wird, ist diese Proportionalität hergestellt. Als Matrixmultiplikation berechnet sich der Fehlervektor E_i aus dem Fehlervektor E_j und den verbindenden Gewichten w_{ji} wie folgt:

$$E_i = W^T \cdot E_j$$

$$\begin{pmatrix} e_{i_1} \\ \vdots \\ e_{i_n} \end{pmatrix} = \begin{pmatrix} w_{11} & \ldots & w_{m1} \\ \vdots & \ddots & \vdots \\ w_{1n} & \ldots & w_{mn} \end{pmatrix} \cdot \begin{pmatrix} e_{j_1} \\ \vdots \\ e_{j_m} \end{pmatrix}$$

Im Gegensatz zur Vorwärtsberechnung, bei der die Summe aller zu j_1 führenden Gewichte multipliziert mit allen $a_{i_1}, \ldots, a_{i_m}$ gesucht wird, multipliziert man bei der Fehlerberechnung die Summe aller zu e_{i_1} führenden Gewichte mit allen $e_{j_1}, \ldots, e_{j_m}$. Dazu nutzt man die transponierte Gewichtsmatrix W^T.

Der so erhaltene Fehlervektor kann nach dem gleichen Prinzip für alle versteckte Schichten berechnet werden.

Gradientenabstiegsverfahren Die Gewichte können nicht so angepasst werden, dass der resultierende Fehler 0 wird. Deshalb wird versucht in mehreren Schritten die Gewichte so zu verändern, dass der Fehler gegen 0 konvergiert.

Der Gesamtfehler ist die Summe der Einzelfehler. Damit sich bei der Aufsummierung positive und negative Fehler nicht aufheben, wird die quadratische Kostenfunktion verwendet. Sie berechnet den Kostenvektor wie folgt:

$$C = (A - Y)^2$$

Im Gegensatz zum Absolutwert weist diese Funktion einen immer kleiner werdenden Gradienten auf, je weiter man sich einem Minimum nähert.

Um zu wissen, ob ein Gewicht w zu groß oder zu klein ist, betrachtet man die partielle Ableitung von $\frac{\partial c}{\partial w}$. Sie gibt an, um wie viel sich c bei einer kleinen Veränderung von w ändert und ist damit der Anstieg der Kostenfunktion.

Durch Ableiten der Kostenfunktion und der Aktivierungsfunktion, kann unter Anwendung der Kettenregel $\frac{\partial c}{\partial w}$ berechnet werden. Ist dieser Anstieg positiv, muss das Gewicht verringert werden, um die Kosten zu senken. Bei einem negativen Wert, minimiert eine Erhöhung des Gewichts die Kosten. Die Anpassung muss also in die entgegengesetzte Richtung zum Anstieg der Kostenfunktion stattfinden. Bei einer proportionalen Anpassung ist die Schrittweite abhängig vom Anstieg. Die Kostenfunktion weist eine Parabelform auf. Nährt man sich dem Minimum, wird der Anstieg kleiner. Deshalb ist die Schrittweite zu Beginn groß und sinkt allmählich. So wird das Überspringen des Minimums durch zu große Schritte verhindert.

$$w' = w - \eta \frac{\partial c}{\partial w}$$

Als Proportionalitätsfaktor wird zusätzlich die Lernrate η verwendet um die Schrittweite zu skalieren. Sie sollte dabei weder zu klein noch zu groß gewählt werden, da das Netz ansonsten zu lange zum Trainieren braucht, oder aber das gefundene Minimum der Kostenfunktion übersprungen wird.

Trainingsverfahren Werden die Gewichte nach jedem Trainingsbeispiel angepasst, sprich man vom Online-Lernen. Es betrachtet beim Wählen des nächsten Schrittes nur eine einzelne Eingabe.

Beim Offline-Lernen werden zunächst alle Trainingsbeispiele durchgegangen und dann die durchschnittliche Gewichtsanpassung ermittelt und angewendet. Dies ist wesentlich zeitaufwändiger, berücksichtigt aber die optimale Veränderung für alle Trainingseingaben.

Gewichtsinitialisierung Zu Beginn des Trainings werden alle Gewichte mit zufälligen Werten initialisiert. Die Initialisierung ist dafür verantwortlich, welche Ausgangswerte das Netz verwendet.

Durch eine ungünstige Ausgangskonfiguration des Netzes kommt es vor, dass Trainingsdurchläufe negativ beeinflusst werden. Es wird dadurch nur ein lokales Minimum der Kostenfunktion gefunden. Um optimale Ausgangswerte für eine Netzkonfiguration zu finden, sollte man mehrere Trainings mit unterschiedlichen Belegungen der Gewichtswerte durchführen. Um zu erreichen, dass die Netzeingabe z in einem definierten Bereich liegt, kann man die Gewichtsinitialisierung abhängig von der Anzahl n der Neuronen der vorherigen Schicht machen.

$$-\frac{1}{\sqrt{n}} < w < \frac{1}{\sqrt{n}}$$

2.6 Trainingsdaten

Um das künstliche neuronale Netz zu trainieren, wird eine große Menge nach Emotionen klassifizierter Bilder von Gesichtsausdrücken benötigt. Diese Bilder sollten ein großes Spektrum unterschiedlicher Gesichtstypen abdecken, damit das trainierte Netz unabhängig von Alter, Hautfarbe, Geschlecht oder Frisur einsetzbar ist. In den drei verwendeten Datenbanken wurden Personen angewiesen eine bestimmte Emotion zu zeigen.

Aus der JAFFE Datenbank werden 183, aus COHN 326 und aus KDEF 840 Bilder verwendet. Idealerweise sollten 70% der verfügbaren Beispiele zum Training und 30% zum Testen genutzt werden. Aufgrund der relativ wenigen Daten wurden die ersten 150 Bilder jeder Emotion zum Trainieren verwendet und die Übrigen zum Testen genutzt. Dies beeinflusst die Testergebnisse, nutzt aber möglichst viele Trainingsdaten. Das Trennen von Trainings- und Testdaten ist wichtig, damit das Netz die Beispiele nicht auswendig lernt (overfitting), sondern allgemeine Strukturen erlernt.

Die Gesichter wurden mit der in 2.3.4 beschriebenen Methode erkannt, auf 100×100 Pixel skaliert und damit normiert. In einer Datenbank wird der Dateiname und der zugeordnete Emotionscode festgehalten. Die Emotionen sind in den genutzten Emotionsdatenbanken zu gleichen Teilen repräsentiert. Allerdings wurden nicht alle Gesichter der Beispieldatensätze erkannt, wodurch es minimale Abweichungen in der Anzahl der einzelnen Emotionen gibt. Die Emotion *verachtend* ist nur in der COHN Datenbank enthalten und deshalb mit 25 (statt 190 bis 250) Bildern unterrepräsentiert.

Abbildung 8: Beispiele aus JAFFE[10], COHN[11] und KDEF[12]

3 Ergebnisse

Die Lernrate, Blockgröße und Struktur der versteckten Schichten sind *Hyperparameter*. Sie werden vorgegeben und empirisch ermittelt. Dabei muss immer ein Kompromiss zwischen höherer Genauigkeit und längeren Trainingszeiten gefunden werden.

Aus einer kleineren Blockgröße resultieren beispielsweise mehr Blöcke und damit aussagekräftigere statistische Daten. Die höhere Blockanzahl erfordert jedoch mehr Eingabeneuronen für das Netz. Daraus ergibt sich eine größere Menge an Gewichten in der Folgeschicht.

Beim Lernen steigt mit zunehmender Gewichtsanzahl der Rechenaufwand. Daher muss eine Blockgröße gewählt werden, welche maximal groß ist um alle Details gerade noch zu beinhalten, jedoch gleichzeitig die Blockanzahl möglichst gering hält.

Die beste Konfiguration erkannte 82,75% der Testbilder richtig. Die Blockgröße ist 10, die Schichten sind (5900, 2000, 7) und die Lernrate beträgt 0.2. In Abbildung 9 sieht man den Kostenverlauf während des Trainings über 10 Epochen (blau) und beim Testen (orange). Da die Kosten bis zum Ende mit Ausschlägen sinken, ist es wahrscheinlich, dass durch weitere Epochen Training noch ein geringerer Fehler zu erreichen wäre. In Abbildung 10 sieht man die Emotionen, die das Netz in einem Auszug aus den Testdaten erkannt hat. Der Farbige Anteil des Balken gibt den Anteil des jeweiligen Ausgabeneurons an der höchsten Ausgabe an.

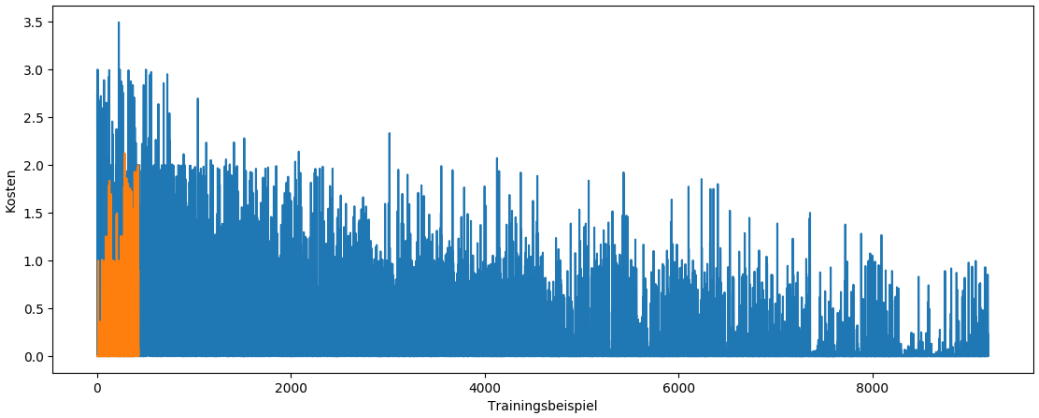


Abbildung 9: Kosten beim Training (blau) sowie beim Testen (orange)

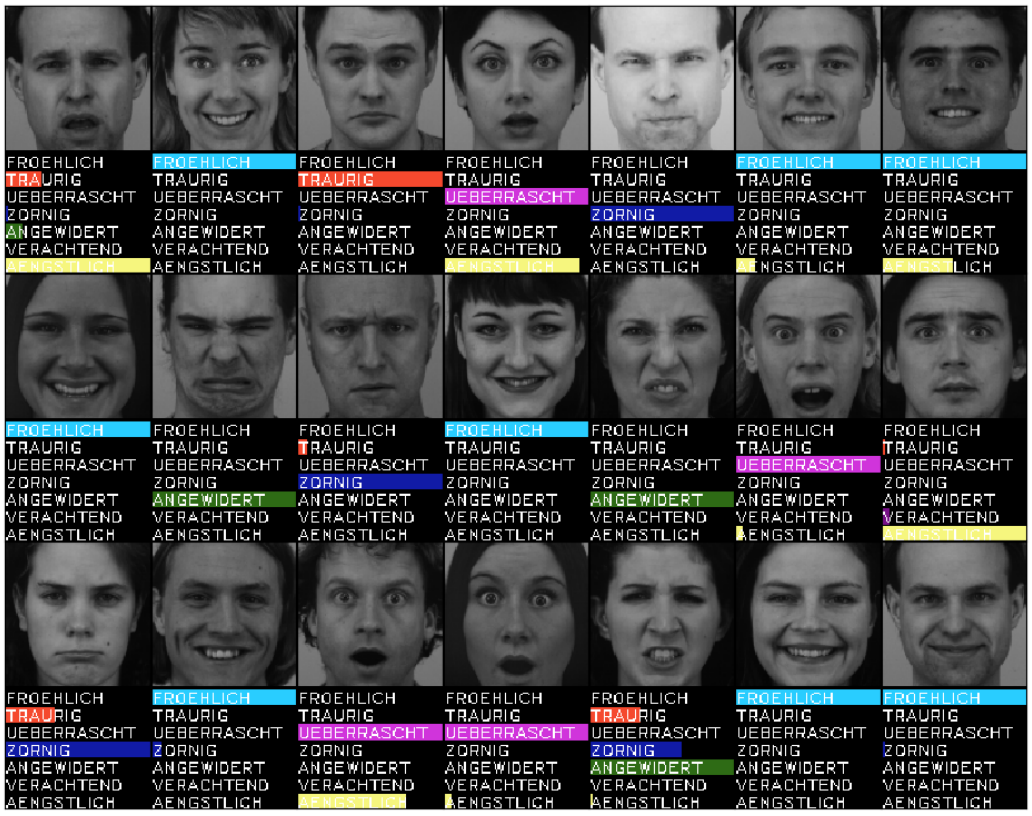


Abbildung 10: Erkannte Emotionen von Testdaten

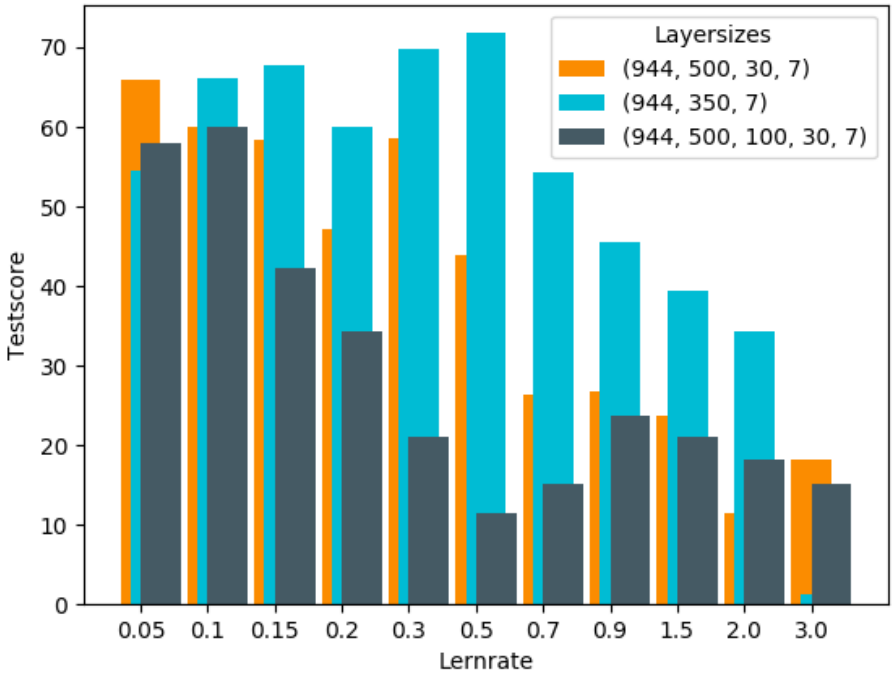


Abbildung 11: Testscores bei verschiedenen Lernraten für unterschiedliche Netzkonfigurationen (siehe Legende) mit Blockgröße 25

In Abbildung 11 sind drei Netzkonfigurationen (orange, blau, schwarz) und deren Testergebnisse nach dem Trainieren mit Lernraten von *0,05; 0,1; 0,15; 0,2; 0,5; 0,7; 0,9; 1,5; 2* und *3* dargestellt. Die Netzkonfiguration mit den geringsten Schichten erzielte immer außer bei einer Lernrate von 0.05 die besten Ergebnisse. Ihre optimale Lernrate liegt bei 0.5, während diese Rate für die Konfiguration mit den meisten Schichten die schlechteste ist.

3.1 Verbesserungen

Mögliche Verbesserungen wären das Ausprobieren von weiteren Netzkonfigurationen mit Schichtanzahl und Schichtgröße. Die Blockgrößen des Local Binary Pattern Verfahrens und die Lernrate bieten auch noch weitere Testmöglichkeiten an. Außerdem würden größere Gesichtsdatenbanken mehr Testdaten ermöglichen und damit die ungleichmäßige Verteilung der Testbilder pro Emotion verhindern, sowie generell mehr Trainingsdaten bereitstellen.

Das Anpassen der Bildhelligkeit oder eine Erhöhung des Kontrasts vor der statistischen Verarbeitung könnten ebenfalls für bessere Ausgangsdaten sorgen.

Zuletzt ist die Beschränkung der Klassifizierung auf genau eine Emotion unrealistisch, da mehrere Emotionen überlagert ausgedrückt werden können. Diese Zweitemotionen werden aber immer als Fehler gewertet.

Das Lernverfahren kann zu einem Offline oder Mini-Batch Verfahren erweitert werden, was präzisere Schritte im Gradientenabstiegsverfahren ermöglicht. Außerdem könnte die Momentummethode eingesetzt werden, um gefundene lokale Minima zu verlassen und weiter nach dem globalen Minimum zu suchen.

4 Anhang

Literatur

[14] Michael A. Nielsen. *Neural Networks and Deep Learning*. Determination Press, 2015. URL: http://neuralnetworksanddeeplearning.com/.

[15] Tariq Rashid. *Neuronale Netze selbst programmieren*. 1. Aufl. O'REILLY, 2017. ISBN: 978-3-96009-043-4.

Websites

[1] *Facial Action Coding System*. Wikipedia. URL: https://de.wikipedia.org/wiki/Facial_Action_Coding_System (besucht am 15. 05. 2018).

[2] *Fueling Next Generation Vehicles with Affectiva Automotive AI*. URL: http://go.affectiva.com/auto (besucht am 26. 04. 2018).

[3] *Camera Module*. URL: https://www.raspberrypi.org/documentation/hardware/camera/README.md (besucht am 20. 05. 2018).

[4] *Color conversions*. OpenCV. URL: https://docs.opencv.org/3.1.0/de/d25/imgproc_color_conversions.html (besucht am 26. 04. 2018).

[5] *Grauwert*. Wikipedia. URL: https://de.wikipedia.org/wiki/Grauwert (besucht am 26. 04. 2018).

[6] Paul Viola und Michael Jones. *Rapid Object Detection using a Boosted Cascade of Simple Features*. URL: http://www.cs.cmu.edu/~efros/courses/LBMV07/Papers/viola-cvpr-01.pdf (besucht am 26. 04. 2018).

[7] *Local binary patterns*. Wikipedia. URL: https://en.wikipedia.org/wiki/Local_binary_patterns (besucht am 01. 05. 2018).

[8] David Kriesel. *Ein kleiner Überblick über Neuronale Netze*. URL: http://www.dkriesel.com/_media/science/neuronalenetze-de-zeta2-2col-dkrieselcom.pdf (besucht am 29. 05. 2018).

[16] *Selbstständigkeitserklärung (Uni Parderborn)*. URL: http://www.uni-paderborn.de/fileadmin/muwi/PDFs/Selbstsa__ndigkeitserkla__rung.pdf (besucht am 06. 06. 2018).

Datenbanken

[10] Michael J. Lyons u. a. *Coding Facial Expressions with Gabor Wavelets, 3rd IEEE International Conference on Automatic Face and Gesture Recognition, pp. 200-205.* 1998.

[11] T. Kanade, J. F. Cohn und Y. Tian. *Comprehensive database for facial expression analysis. Proceedings of the Fourth IEEE International Conference on Automatic Face and Gesture Recognition (FG'00), Grenoble, France, 46-53.* 2000.

[12] D. E. Lundqvist, A. Flykt und A. Öhman. *The Karolinska Directed Emotional Faces - KDEF, CD ROM.* Department of Clinical Neuroscience, Psychology section, Karolinska Institutet, 1998. ISBN: 9163071649.

[13] P. Lucey u. a. *The Extended Cohn-Kanade Dataset (CK+): A complete expression dataset for action unit and emotion-specified expression. Proceedings of the Third International Workshop on CVPR for Human Communicative Behavior Analysis (CVPR4HB 2010), San Francisco, USA, 94-101.* 2010.

Bild

[9] Qef. *The logistic curve.* URL: `https://en.wikipedia.org/wiki/Sigmoid_function#/media/File:Logistic-curve.svg` (besucht am 30. 05. 2018).

Der Quellcode, welcher für diese Arbeit entstand ist unter GPLv3 veröffentlicht: `https://github.com/hermlon/nnemotions`

Selbstständigkeitserklärung Hiermit erkläre ich, dass ich die vorliegende Arbeit selbstständig und ohne unerlaubte fremde Hilfe angefertigt, keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und die den verwendeten Quellen und Hilfsmitteln entnommenen Stellen als solche kenntlich gemacht habe.[16]

Halle (Saale)
2018-06-07