Projektarbeit: Schere, Stein und unser Paper - Deep Learning und Objekterkennung

Autoren: Lennart Wallis[1], Christian Schneider[2], Marvin Kraus[3]
Supervisor: Prof. Dr.-Ing. Jan Salmen

Zusammenfassung
Im Rahmen einer Projektarbeit des Moduls *Deep Learning und Objekterkennung* der TH Köln wurde mit einem NN eine Klassifikation von Echtbildern realisiert. Thematisch sollte eine Klassifikation der bekannten Handzeichen Schere, Stein und Papier erfolgen, welche um eine weitere Klasse für nicht zulässige Handzeichen ergänzt wurde. Ziel dieser Arbeit ist es, den Inhalt des Moduls iterativ in das Modell einfließen zu lassen, und so das Modell immer weiter zu verbessern.

Nach Einführung von *Early Stopping* wurde die LR, der *Dropout* sowie die Netzstruktur systematisch angepasst, um eine möglichst gute Hyperparameterkonfiguration für das NN zu finden. Es konnte gezeigt werden, dass durch eine iterative Anpassung der Hyperparameter eine Genauigkeit von 97.6% auf dem Trainingdatensatz erreicht und damit unser Erwartungswert von 95% überschritten werden konnte. Auf dem Validierungsdatensatz betrug die Genauigkeit 92.9%.

Zusätzlich wurde eine neue, alternative Fehlermetrik mit reduzierter Klassenanzahl implementiert, welche jedoch eine geringere Genauigkeit als erhofft aufwies. Abschließend wurden Fehlerquellen erörtert und entsprechend Optimierungsmaßnahmen aufgezeigt.

[1] *lennart.wallis@smail.th-koeln.de*
[2] *christian_michael.schneider@smail.th-koeln.de*
[3] *marvin.kraus@smail.th-koeln.de*

Inhaltsverzeichnis

1. Projektvorstellung

Im Rahmen eines Hochschulprojektes des Fachs *Deep Learning und Objekterkennung* sollen die in diesem Modul vorgestellten Methoden zur Entwicklung von Deep Learning an einem Fallbeispiel umgesetzt werden. Hierzu soll ein Neuronales Netzwerk (NN) eine Bildklassifikation über Handzeichen des bekannten Spiels Schere, Stein, Papier umsetzten. Dieses wird durch eine vierte Klasse *undefined* erweitert, welche ungültige Handzeichen beinhaltet. Es wurde ein Trainingsdatensatz der Plattform *Kaggle* ausgewählt, ergänzt durch weitere Bilder der StudentInnen.

1.1 Erwartung an das Project & State of the Art

Auf dem Basisdatensatz von *Kaggle* gibt es mehrere Projekte, die eine Genauigkeit von 99% bei der Klassifikation aufweisen (Siehe [6]). Demnach ist zu erwarten, dass dieses Projekt verhältnismäßig gute Ergebnisse erzielen kann (ca. 90-95%). Der Erwartungswert ist bewusst kleiner gewählt, da:
1. durch die neu zum Datensatz hinzu gekommenen Bilder der Klassen Schere, Stein und Papier die Aufgabe für das NN schwerer geworden ist (siehe Kapitel Der Datensatz); 2. durch das Hinzufügen der vierten Klasse und der zu dieser zugehörigen Bilder die Komplexität erhöht wird (siehe Kapitel Der Datensatz).

1.2 Der Datensatz

Der Datensatz besteht aus 2 Teilen. Der 1. Teil ist der auf der Internetplattform *Kaggle* veröffentlichte Datensatz, zu finden unter [2]. Dieser besteht aus 2188 Bildern für die Klassen Schere, Stein und Papier. Die Bilder zeigen jeweils eine Hand von verschiedenen Personen über einem grünem Hintergrund die die Gesten für Schere, Stein und Papier zeigen. Ein typisches Bild ist in Hand aus dem Kaggle Datensatz zu sehen.

Abbildung 1. Hand aus dem Kaggle Datensatz

Der 2. Teil besteht aus Aufnahmen der StudentInnen. Diese haben ebenfalls Bilder zu den Klassen Schere, Stein und Papier aufgenommen. Diese haben jedoch keinen einheitlich grünen Hintergrund, haben verschiedene Auflösungen und weisen auch sonst eine deutlich höhere Varianz auf. Diese kommt daher, dass viele unterschiedliche Hände in unterschiedlichen Haltungen vorkommen, die Perspektive verschieden ist und Ärmel oder Accessoires zu sehen sein können. Zusätzlich ist es erlaubt, dass - anders als im Datensatz von *Kaggle* - Hände auch von links in Bild ragen. Eine weiter Schwierigkeit bildet die Erweiterung um die Klasse *undefined*. Ein Bild gehört dann zu dieser Klasse, wenn es kein zulässiges der drei Handzeichen abbildet. Das NN soll darauf trainiert werden, hiermit entsprechend umzugehen und diese Klasse ebenfalls korrekt zu erkennen. Der Datensatz ist folgendermaßen aufgebaut:

	Schere	Stein	Papier	Undefined
Kaggle	750	726	712	0
StudentInnen	526	523	553	660
Gesamt	1276	1249	1265	660

Die Anzahl der Bilder innerhalb der Klassen ist nicht ausgeglichen. Die Abweichung Klassen Schere, Stein und Papier sind kleiner als 3%. Am größten ist der Unterschied zu der Klasse *undefined*, welche weniger als halb so oft vertreten ist.

Die ungleiche Klassenverteilung hat eine Auswirkung auf die Genauigkeit des Netzes, falls die Daten in der Wirklichkeit eine andere Verteilung haben. Da wir davon ausgehen, dass dies jedoch nicht der Fall ist, gehen wir im Folgenden nicht weiter darauf ein.

2. Das Basis Modell

Um die Auswirkung der einzelnen Teilschritte erläutern zu können, ist diese Arbeit iterativ aufgebaut. Hierzu wurde ein Basismodel festgelegt, das den Startpunkt aller weiteren Erläuterungen bilden soll und in Einzelschritten immer weiter angepasst wird. In allen folgenden Modellen wurde mit *Cross-Validation* und einer 80 / 20 Spaltung trainiert und validiert.

Das Basismodel wurde über 20 Epochen auf dem nicht augmentierten Datensatz mit einer Lernrate (LR) von 0.0005 (siehe Kapitel Anpassung der Lernrate) und einer *Batch Size* von 32 trainiert. Als *Optimizer* kam *Adam* zum Einsatz, siehe Kapitel Anpassung der Lernrate.

Um zu verhindern, dass wir für unser Basismodel viele Testläufe benötigen, bis wir ein halbwegs gutes Ergebnis reproduzieren können, haben wir uns bei der Netzstruktur an bereits bewährten Netzstrukturen für einen ähnlichen Aufgabentyp orientiert. Demnach haben wir unsere Layerstruktur gleich wie das für den *MNIST* Datensatz ausgelegte *AlexNet* aufgebaut. Im folgenden ist die Performanz Baseline Model - Netz 00 dargestellt:

Ein Überblick über die iterative Anpassung der Hyperparameter kann der Tabelle 1 entommen werden. Zur Orientierung werden folgend *Netznummern* ausgewiesen.

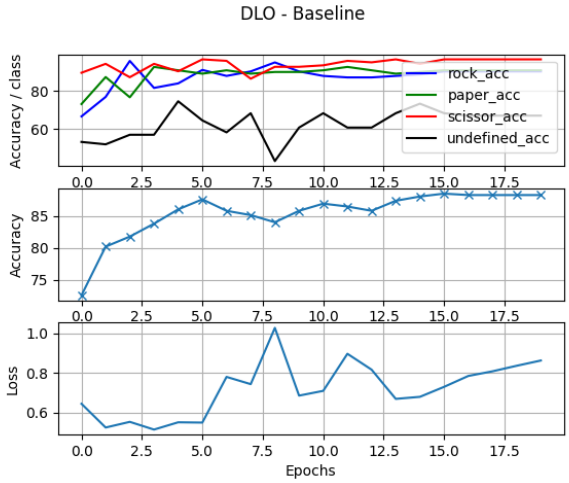

Abbildung 2. Performanz Baseline Model - Netz 00

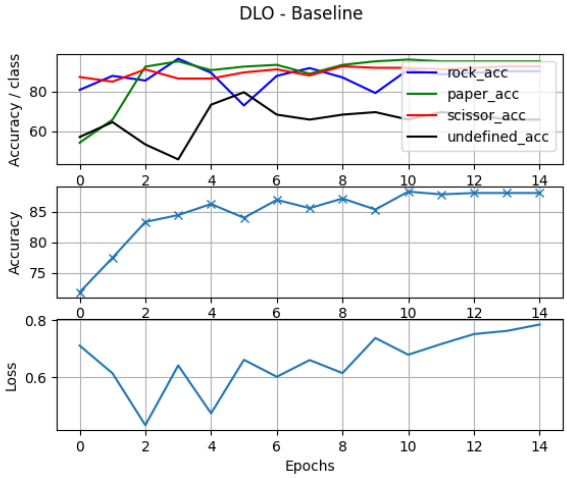

Abbildung 3. Early Stopping - Netz 01

2.1 Performanz des Basis Modells

Das Basis Modell erreicht mit den im Kapitel Das Basis Modell beschriebenen Hyperparametern eine Genauigkeit von fast 90%. Betrachtet man die Genauigkeit pro Klasse (siehe Performanz Baseline Model - Netz 00) wird deutlich, dass das Netz bei der *undefined*-Klasse die geringste Genauigkeit hat, ca. 20% weniger als bei den Klassen von Schere, Stein oder Papier.

3. Optimierung des Basis Modells (*AlexNet*)

Die nächsten Schritte zur Implementierung sind *Early Stopping*, eine Anpassung der *Lernrate* und falls nötig das Hinzufügen von *Dropout*, um die Performanz unseres Basis Modells zu erhöhen.

3.1 Implementierung Early Stopping

Sollte beim Training des Netzes eine Überanpassung auftreten, ist es notwendig den Trainingsprozess an dieser Stelle abzubrechen. Aus diesem Grund haben wir eine *Early Stopping* Methode in unsere Algorithmik eingebaut.

Das Modell bricht das Training ab, sobald der *Loss* im Verlauf von 4 Epochen im Vergleich zur jeweils vorausgegangenen Epoche weiter ansteigt (wie z.B. zwischen Epoche 10 bis 14 zu sehen ist). Dies bringt jedoch das Problem mit sich, dass kein Abbruch erfolgt, falls der *Loss* innerhalb von 4 Epochen fluktuiert. Dies kann z.B. zwischen den Epochen 2-10 beobachtet werden. Jedoch haben wir die Chance für das Auftreten dieses Falles als gering genug eingeschätzt. Durch den Verlauf des *Losses* über die Epochen können wir solche Schwankungen ausfindig machen.

Eine mögliche Alternativen ist eine komplexere Auswertungsmethodik, die zusätzlich noch das Verhältnis aus Anstiegen und Abstiegen des *Loss* betrachtet. Dies erschien uns jedoch nicht als nötig, weswegen wir bei der simplen Methodik geblieben sind (welche sich auch für den gesamten Verlauf des Projektes bewährt hat).

3.2 Dropout

Über den Einsatz von *Dropout* werden die Gewichte einzelner Neuronen des NN immer wieder auf 0 gesetzt. Dadurch wird erreicht, dass das NN nicht von einzelnen Werten bestimmter Neuronen abhängig ist, sondern auch andere Neuronen diese Fälle abdecken können. Dieser Prozess ist eine beliebte Technik, um eine Überanpassung des Netzes auf die Trainingsdaten zu verringern.

Um die Auswirkung des *Dropouts* auf die Performanz des Netzwerks zu messen haben wir drei unterschiedliche Messungen ausgeführt. Wir haben uns aufgrund der Empfehlungen aus [1] einen *Dropout* von 0.4, 0.5 und 0.6 getestet.

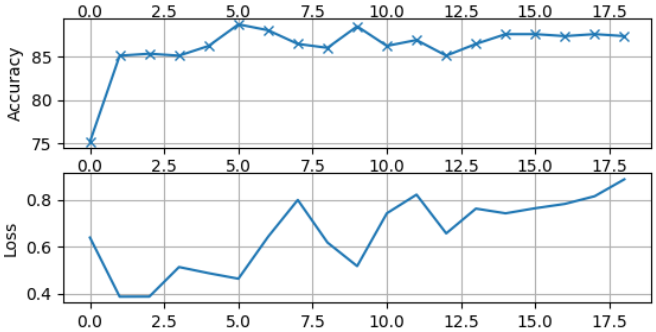

Abbildung 4. Baseline: Anpassung Dropout: 0.4 - Netz 02

Grundsätzlich würde man erwarten, dass bei einem höheren *Dropout* mehr Epochen benötigt werden, bis sich eine Konvergenz eingestellt hat, da man durch das zufällige Aussetzen einzelner Neuronen Schwierigkeiten in das Netz einträgt, dessen Bewältigung das Netz lernen soll. Diesen Effekt kann man zwischen 0.4 und 0.6 deutlich beobachten, hier liegen die Konvergenzen 5 Epochen auseinander (ca. 13 bei 0.4 und 18 bei 0.6). Bei einem *Dropout* von 0.5 wird der Trainingspro-

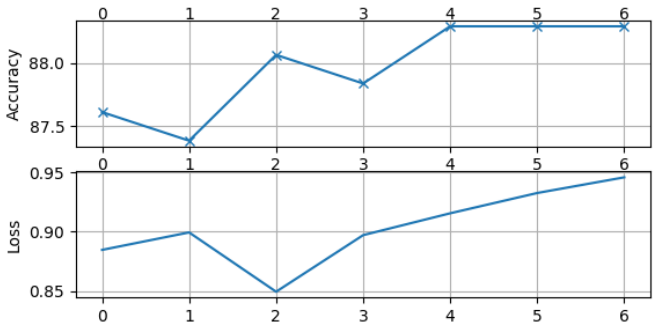

Abbildung 5. Baseline: Anpassung Dropout: 0.5 - Netz 02

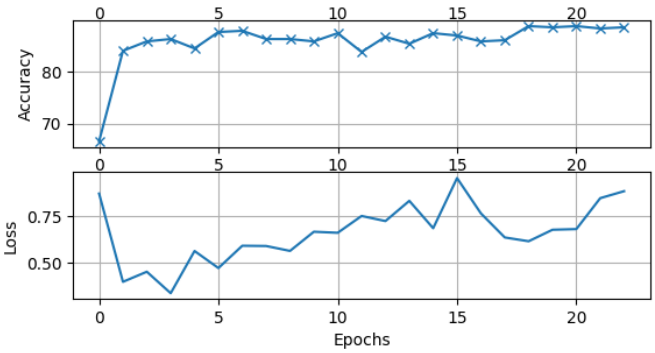

Abbildung 6. Baseline: Anpassung Dropout: 0.6 - Netz 02

zess durch unserer *early stopping* Methodik abgebrochen und hat aus diesem Grund verhältnismäßig wenig Aussagekraft. Da 0.6 die höchste Genauigkeit erzielt hat, während der *Loss* bei allen in der gleichen Größenordnung lag haben wir uns für die Wahl dieses Hyperparameters entschieden.

3.3 Anpassung der Lernrate

Die LR ist einer der wichtigsten Hyperparameter für ein NN. Wählt man die LR zu klein kann es (zu) lange dauern bis es zu einer Konvergenz kommt. Anderseits darf man die LR auch nicht zu groß einstellen, da es ansonsten möglicherweise für das NN nicht möglich ist, zu konvergieren, da es immer über das Ziel hinausschießt und anschließend wieder untersteuert. Als Grundsatz gilt es hierbei die LR größtmöglich zu wählen, aber nicht zu groß, sonst kann das Netz nicht mehr entsprechend konvergieren. Empfohlen wird hierbei ein Start bei einer kleinen LR bis hin zu einer großen. Unsere Werte und Herangehensweise folgt dabei der Empfehlung aus: [3] Seite: 325-326.

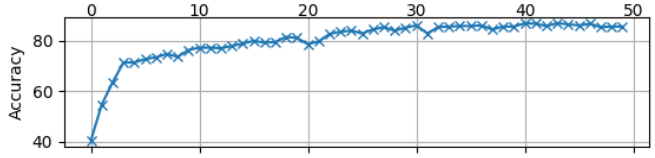

Abbildung 7. Baseline: Anpassung der LR: 0.00001 - Netz 03

Wie aus Baseline: Anpassung der LR: 0.0001 - Netz 03 ersichtlich hat eine LR von 0.0001 die höchste Genauigkeit

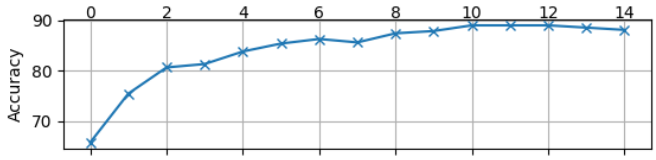

Abbildung 8. Baseline: Anpassung der LR: 0.0001 - Netz 03

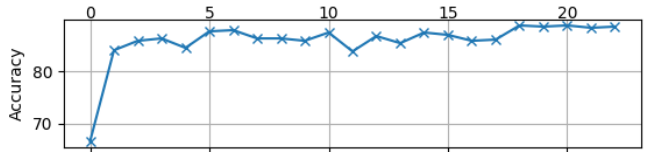

Abbildung 9. Baseline: Anpassung der LR: 0.0005 - Netz 03

von ca. 88% erzielt. Höhere Werte haben sich als nicht so genau erwiesen. Aus diesem Grund haben wir uns im folgenden für die Festlegung dieses Wertes für die LR entschieden.

Um den Vorgang des Trainings besonders in späteren Epochen zu verbessern ist es hilfreich die LR auf den Trainingsfortschritt anzupassen. Hierfür kommen sogenannte *Optimizer* zum Einsatz, welche die LR im Laufe des Trainings verkleinern. Hier haben wir uns für den *Adam Optimizer* aufgrund der Empfehlung aus [4] entschieden. Die Messergebnisse sind hierbei jedoch mit Vorsicht zu genießen, da der *Optimizer* einen starken Einfluss auf die LR hat und somit möglicherweise unsere Testergebnisse verzerrt. Diesbezüglich sollte man in zukünftigen Projekten den *Optimizer* erst initialisieren nachdem man die optimale LR ermittelt hat. Dennoch sollte unser Testergebnis in der richtigen Größenordnung liegen.

4. Data Augmentation

Um unser NN robuster gegenüber Varianzen, also unterschiedlichen Darstellungsformen, in den Daten zu machen, haben wir aus den gegebenen Trainingsbildern neue Trainingsbilder generiert.

Wir vermuten, dass durch *Data Augmentation* eine Erhöhung der Genauigkeit erfolgt, da die einzelnen Trainingsbilder sehr unterschiedlich zueinander sind. Beispielsweise haben die Trainingsbilder sehr unterschiedliche Hintergründe (in Farbe und Struktur) und möchten Effekte verhindern wie z.b. dass das NN möglicherweise falsche Schlüsse aufgrund der Hintergrundfarbe schließt.

Im folgenden stellen wir die einzelnen Teilschritte der *Data Augmentation* vor. Abschließend folgt eine Analyse des Einflusses der *Data Augmentation* auf die Genauigkeit. Alle Befehle können in der PyTorch Dokumentation nachgelesen werden ([5]). Ein nicht bearbeitetes Eingabebild ist in Hand Originalbild zu sehen.

```
transforms.Resize([128, 128])
```

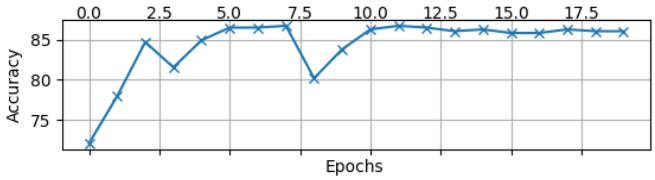

Abbildung 10. Baseline: Anpassung der LR: 0.001 - Netz 03

Abbildung 11. Hand Originalbild

Das NN erwartet die Eingabebilder immer in der gleichen Größe. Wir haben uns für 128x128 Pixel entschieden, anstatt der 300x200 Pixel in denen die Bilder im Original vorliegen. Dies hat verschiedene Gründe:
1. Alle Modelle aus der PyTorch Bibliothek verwenden quadratische Eingabebilder.
2. Eine Reduktion von Pixeln reduziert die Anzahl der benötigten Neuronen im NN und erhöht somit die Trainingsgeschwindigkeit.
3. Die Bilder sind immer noch sehr gut mit dem Auge erkennbar.
Dieser Schritt ist eigentlich kein Teil der Datenaugmentierung sondern der Preprocessing Pipeline, ist jedoch notwendig damit das NN die Daten verarbeiten kann. Ein skaliertes Bild ist in Hand skaliert auf 128x128 Pixel abgebildet.

Abbildung 12. Hand skaliert auf 128x128 Pixel

```
transforms.ColorJitter()
```

ColorJitter verändert die Helligkeit, den Kontrast, die Sättigung und den Farbton eines Bildes um zufällige Werte. Hierdurch wollen wir eine Robustheit gegenüber verschiedenen Kameras und Aufnahmebedingungen (Licht an, Tageslicht, Nacht etc.) erreichen.

Abbildung 13. Hand skaliert auf 128x128 Pixel und ColorJitter

```
transforms.GaussianBlur(kernel_size=3)
```

Gaussian Blur ist ein Filter, der das Bild verwäscht, d.h. dass ein Pixel mit den gewichteten Pixeln aus seiner Nachbarschaft (abhängig von der Größe des Kerns) zusammen addiert wird. Der neue Pixel ist also von seinen Nachbarpixeln beeinflusst. Dies dient dazu, dass das NN nicht mehr auf bestimmte Kanten trainieren kann und soll somit zu einem robusteren Netz beitragen. Zudem wird rauschen welches eventuell durch Kameraaufnahmen entsteht, unterdrückt.

Abbildung 14. Hand skaliert auf 128x128 Pixel, ColorJitter und GaussianBlur

```
transforms.RandomRotation(degrees=20)
```

Random Rotation dreht die Bilder zufällig in einem Winkel zwischen -20 und 20 Grad. Hierbei wurde darauf geachtet, dass die Bilder nicht zu stark gedreht werden, da hierdurch
1. Schwarze Flächen entstehen und
2. Hände nur von links und rechts, nicht von oben und unten in das Bild hinein ragen.

```
transforms.RandomVerticalFlip()
```

Abbildung 15. Hand skaliert auf 128x128 Pixel, ColorJitter, GaussianBlur und RandomRotation

```
transforms.RandomHorizontalFlip()
```

Über *RandomVerticalFlip* und *RandomHorizontalFlip* wurden die Bilder entsprechend horizontal bzw. vertikal gespiegelt. Dies hat den Effekt, dass Hände die von rechts in das Bild ragen auch von links in das Bild ragen können bzw. dass eine rechte Hand auch eine linke Hand sein kann. Dies ist besonders hilfreich, da dieser Freiheitsgrad durch die StudentInnen hinzugefügt wurde und dieser durch die Augmentation auf alle Bilder erweitert wird.

Abbildung 16. Hand skaliert auf 128x128 Pixel, ColorJitter, GaussianBlur, RandomRotation und RandomVerticalFlip und RandomHorizontalFlip

```
transforms.Normalize((0.5), (0.5))

transforms.ToTensor()
```

Im Gegensatz zu unserer ersten Annahme, normalisiert *.Normalize(mean, std)* nicht sondern subtrahiert von jedem Wert 0.5 und teilt anschließend jeden Wert durch 0.5. Die Funktion ToTensor() hingegen skaliert alle Werte vom Intervall [0, 255] auf das Intervall [0, 1].

```
transforms.Grayscale()
```

In den Farben stecken keine wichtigen Informationen für das NN. Wichtig sind zusammenhängende Bereiche von Pixeln (sog. Segmente), wie z.B. die Hand und der Hintergrund. Um nicht unnötige Informationen zu verarbeiten, verwenden wir Grauwertbilder.

Wie bereits angesprochen unterscheiden sich der Basis Datensatz von *Kaggle* und der von den StudentInnen erstellte Datensatz stark in Größe und Komplexität der Dargestellten Formen, aber auch in den Farben. Wir möchten vermeiden, dass das NN falsche Zusammenhänge trainiert, wie z.B. dass kein Bild mit einem grünen Hintergrund zu der Klasse *undefined* gehört.

Abbildung 17. Hand skaliert auf 128x128 Pixel, ColorJitter, GaussianBlur, RandomRotation, RandomVerticalFlip, RandomHorizontalFlip und als Grauwertbild

```
transforms.RandomPerspective()
```

Ein Ändern der Perspektive sollte sich positiv auf die Genauigkeit des Netzes auswirken, da sowohl die Kameraposition als auch die Position der Hand von Bild zu Bild stark variieren können. In unseren Tests hat sich dies aber nicht bewahrheitet, weswegen wir im weiteren diesen Funktionsaufruf auskommentiert gelassen haben. Möglicherweise liegt dies an den neuen Formen die durch die perspektivische Verzerrung entstehen.

4.1 Performanzeinfluss der Augmentation

Da Teile der Augmentation zufällig erfolgen - ColorJitter, HorizontalFlip, VerticalFlip, Perspektive und Rotation - haben wir uns dazu entschieden, aus jedem Eingabebild 12 neue Bilder zu erstellen. Die Augmentation der Daten liefert eine Erhöhung von ca. 88.9 % auf ca. 95.9 % Prozent, also

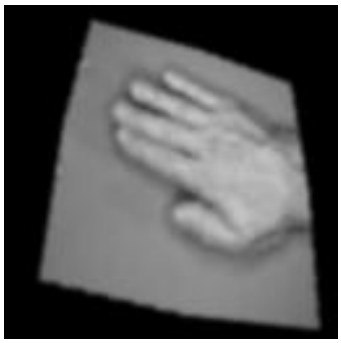

Abbildung 18. RandomPerspective

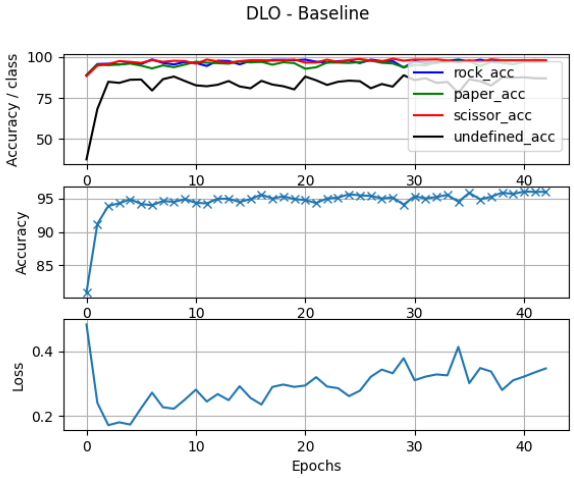

Abbildung 19. Hinzufügung von augmentierten Daten - Netz 04

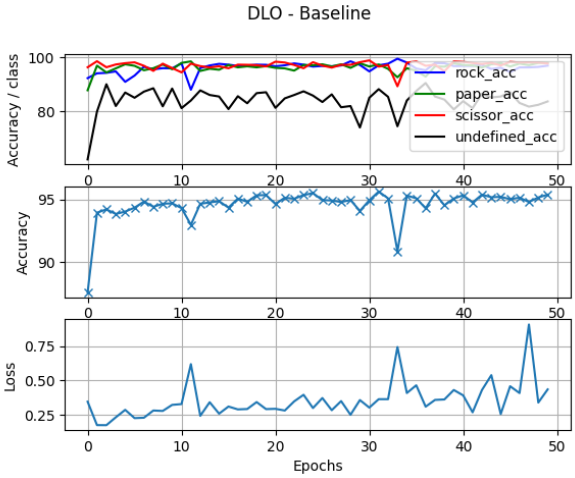

Abbildung 20. Erweiterung der Netzstruktur - Netz 11

eine Verbesserung um ganze 7 Prozentpunkte (vgl. Baseline: Anpassung der LR: 0.0001 - Netz 03 und Hinzufügung von augmentierten Daten - Netz 04).

5. Modifikation AlexNet

Die Komplexität des Netzes, also die Größe der Schichten und deren Anzahl und Verkettung ermöglichen es einem NN komplexe Zusammenhänge abzubilden folglich gilt: je höher diese Komplexität desto komplexere Zusammenhänge kann das Netz abbilden jedoch unter verlängerter Trainingszeit (Epochenanzahl bis zur Konvergenz). Die erhöhte darstellbare Komplexität kann jedoch dazu führen, dass sich das NN zu gut an die Trainingsdaten anpassen kann und entsprechend nicht mehr in der Lage ist zu verallgemeinern. Der Aufbau eines NN spielt demnach eine entscheidende Rolle und soll in dem folgenden Test angepasst werden. Da der Aufbau der Daten komplexer als der *MNIST* Datensatz ist (mehrere Farben, eine komplexere Darstellen sowie mehr Pixel) wollten wir testen ob eine Erweiterung unserer Basisnetzstruktur eine mögliche Verbesserung beinhaltet.

Bei einem direkten Vergleich zwischen Hinzufügung von augmentierten Daten - Netz 04 und Erweiterung der Netzstruktur - Netz 11 ergibt sich keine wirkliche Verbesserung der Genauigkeit bei ansteigendem Loss . Eine Verkleinerung der Netzstruktur brachte hingegen folgendes Ergebnis: Verkleinerung der Netzstruktur - Netz 31

Entgegen unserer Erwartung hat das Netz mit verkleinerter Netzstruktur tatsächlich nochmal 1-2 Prozentpunkte besser abgeschnitten, was in diesem Bereich eine deutliche Verbesserung darstellt. Dazu ist zu sagen, dass wir um die Netzstruktur wesentlich ändern zu können ohne immense Größenunterschiede zwischen den einzelnen Layern zu haben, die Eingangsgröße des Bildes verkleinert haben (64x64).

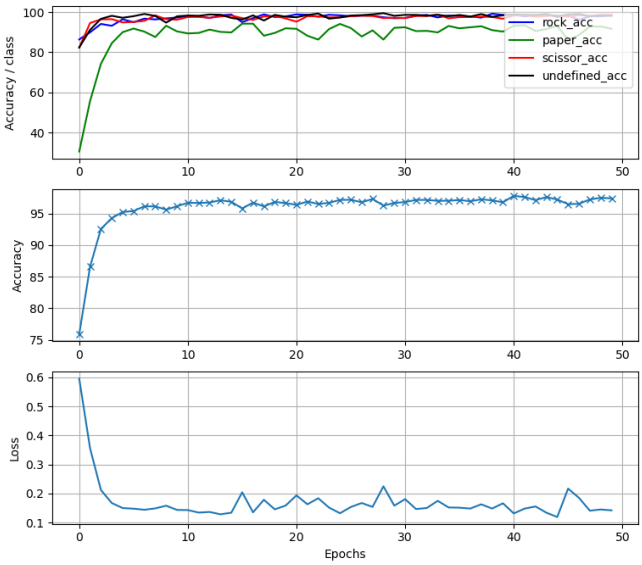

Abbildung 21. Verkleinerung der Netzstruktur - Netz 31

6. Fazit & Ausblick

Durch verschiedenste Anpassungen unseres Basismodells, welches wir an ein bereits erfolgreiches Netz (*AlexNet*) angelehnt haben konnten wir unseren Erwartungswert von 95% erreichen, wie in Verkleinerung der Netzstruktur - Netz 31 abgebildet. Besonders die Augmentation neuer Daten hatte einen großen Einfluss auf die Genauigkeit des NN.

6.1 Test auf dem Validierungsdatensatz

Auf dem Validationdatensatz erreichen wir eine Genauigkeit von ca. 92.9%. Dies sind 4.7 Prozenpunkte weniger als auf dem Testdatensatz, auf dem wir eine Genauigkeit von 97.6% erreicht haben. Somit liegt unser Generalisierungsfehler unter 5%. Dies kann mehrere Ursachen haben: Overfitting, eine Diskrepanz zwischen Test- und Validierungsdatensatz oder ein nicht ausreichend langer Trainingsprozess.

Verkleinerung der Netzstruktur - Netz 31 legt jedoch nah, dass es sich nicht um *Overfitting* handelt, da der *Loss* nicht ansteigt während die Genauigkeit des Netzwerks weiter ansteigt. Hier wären weitere Trainingsepochen möglicherweise aufschlussgebend.

6.2 Outlier elimination

Ein typischer erster Schritt in der Pipeline für Projekte mit NN ist Datenaufbereitung. Beim Durchsichten der Daten fällt auf, dass einige der Bilder aus dem *Kaggle* Datensatz nach unseren Kriterien zur Klasse *undefined* gehören sollten, ein Beispiel dafür ist in Abbildung 22 zu sehen.

Abbildung 22. Ein Bild der Klasse Hand welches nach unseren Kriterien zur Klasse *undefined* gehört

Dies wirkt sich negativ auf die Genauigkeit auf und wäre ein weitere Ansatzpunkt, um diese zu steigern.

6.3 Hyperparameter Anpassung

Um einen größeren Suchraum für Hyperparameter aufzustellen, hätten wir GridSearch oder RandomSearch verwenden können ([3] Seite 76f). Hauptziel von uns war jedoch nicht das genaueste NN zu trainieren, sondern die Auswirkungen der Hyperparameter zu betrachten.

6.4 Neue Fehlermetrik für *undefined*-Klasse

Bei der Genauigkeit der einzelnen Klassen fällt auf, dass die *undefined* Klasse die geringste Genauigkeit aufweist. Um die Genauigkeit des NN insgesamt zu erhöhen haben wir versucht eine neue Fehlermetrik zu etablieren, welche ohne die *undefined* Klasse auskommt. Bei der Auswertung muss für jede Klasse eine Konfidenzgrenze überschritten werden um eine Zugehörigkeit zu einer Klasse feststellen zu können. Sollte für jede Klasse die Konfidenz nicht ausreichend sein wird das Objekt der Klasse *undefined* zugewiesen.

Hierzu wurden die Trainings- und Testdaten der Klasse *undefined* entfernt und das Netz testweise auf die bisher ermittelten optimalen Konfiguration der Hyperparameter trainiert. Die Aktivierungsfunktion des Modells des letzten Layers wurde entsprechend zu einer *Softmax*-Funktion angepasst um die Konfidenz zwischen 0 und 1 abzubilden. Für die Klassifikation wird die Konfidenz der Zuweisungen eines einzelnen Bildes ermittelt und die höchste wird entsprechend als Klasse dem einzelnen Bild zugeordnet. Sollte die Konfidenz für alle drei Klasse einen Grenzwert unterschreitet wird die *undefined* Klasse entsprechend zugewiesen. Nach mehreren Test Durchläufen wurde hierfür ein optimaler Schwellwert von 90% ermittelt.

Dies bietet den Vorteil, dass sich das NN wesentlich schneller trainieren lässt. Jedoch schneidet diese Fehlermetrik wesentlich schlechter als die klassische Klassifikation auf vier Klassen (88%) ab. Die Genauigkeit der *undefined* Klasse fällt jedoch von über 80% auf ca. 20% ab, während die anderen Klassen einen kleineren Verlust im einstelligen %-Bereich wahrnehmen. Aufgrund dessen, das die *undefined* Klasse so gering vertreten ist im Vergleich zu den anderen Klassen hat das keine so starke Auswirkung, in der Realität wäre dieses Verfahren jedoch unbrauchbar. Über weiteres fine-tuning wäre es sicher möglich auch diese Fehlermetrik zu einem sinnvollen Ergebnis zu bringen, ob dies jedoch erfolgsversprechender ist bleibt in Frage zu stellen.

6.5 Ausblick

Jedes Bild besteht idealerweise aus 2 Segmenten: Dem Hintergrund und der Hand im Vordergrund. Interessant für die Klassifikation sollte nur die Hand sein. Es wäre demnach möglich, im einem ersten Schritt eine Segmentation durchzuführen gefolgt von einer Binarisierung, sodass die Hand weiss und der Hintergrund schwarz dargestellt werden.

Es sollte möglich sein den schwarzen Rand weitestgehend zu minimieren (oder sogar zu eliminieren), falls die Augmentationsoperationen auf den 300x200 Pixel großen Bildern ausgeführt wird und anschließend das Bild nicht skaliert wird, sondern beschnitten. Da die Position der Hand jedoch sehr unterschiedlich sein kann und nicht in der Mitte des Bildes liegen muss, ist dies nicht trivial.

Abkürzungsverzeichnis

LR Lernrate

NN Neuronales Netzwerk

Tabellen

Netznummer	**00**	**01**	**02**	**03**	**05**	**11**	**12**	**31**
Input Layer Size	53824	53824	53824	53824	53824	**63084**	63084	**5408**
Hidden Layer Size	1024 256	1024 256	1024 256	1024 256	1024 256	**5000** **3000** **1000** **256**	5000 3000 1000 256	**1024** **256** **128**
Output Layer Size	4	4	4	4	**3**	**4**	**3**	**4**
Epochen	20	**50**	50	50	50	50	50	50
Early Stopping Rate		**4**	4	4	4	4	4	4
Learning Rate	0.0005	0.0005	0.0005	**0.0001**	0.0001	0.0001	0.0001	0.0001
Batch Size	32	32	32	32	32	32	32	32
Dropout Rate			**0.6**	0.6	0.6	0.6	0.6	0.6

Tabelle 1. Übersicht über die Hyperparameter der Netzgenerationen

Literatur

[1] J. Brownlee. A gentle introduction to dropout for regularizing deep neural networks. https://machinelearningmastery.com/dropout-for-regularizing-deep-neural-networks/. Accessed: 26-08-2022.

[2] J. de la Bruère-Terreault. Rock-paper-scissors images. https://www.kaggle.com/datasets/drgfreeman/rockpaperscissors. Accessed: 29-08-2022.

[3] A. Geron. *Hands-on machine learning with Scikit-Learn and TensorFlow : concepts, tools, and techniques to build intelligent systems*. O'Reilly Media, Sebastopol, CA, 2017. ISBN 978-1491962299.

[4] prakharr0y. Intuition of adam optimizer. https://www.geeksforgeeks.org/intuition-of-adam-optimizer/. Accessed: 01-09-2022.

[5] Pytorch. Transforming and augmenting images. http://pytorch.org/vision/main/transforms.html. Accessed: 14-09-2022.

[6] Q. Shaikh. Rock-paper-scissors images project. https://www.kaggle.com/code/quadeer15sh/tf-keras-cnn-99-accuracy. Accessed: 12-09-2022.