Neuronales Netz – Datensatz: MINST

Dokumentation Projekt KI

an der Dualen Hochschule Baden-Württemberg Heidenheim

in der Fakultät Wirtschaft

im Studiengang Wirtschaftsinformatik

eingereicht von

Denis Gebele

Maja Greiner

Sofya Midler

Semester: 2023/5

Abgabetermin: 14.09.2025

Dozent: Dr. Ralf Höchenberger

Inhaltsverzeichnis

Abbildungsverzeichnis

Abkürzungsverzeichnis

Adam ..... *Adaptive Moment Estimation*
CNN ..... *Convolutional Neural Network*
ECE ..... *Expected Calibration Error*
LR ..... *Lernrate*, *Lernrate*
MLP ..... *Multilayer Perceptrons*
ReLU ..... *Rectified Linear Unit*
SGD ..... *Stochastischer Gradientenabstieg*

1 Einleitung

In diesem Projekt wird ein neuronales Netz für den MNIST-Datensatz entwickelt und schrittweise optimiert (vgl. Anhang 1). Zuerst werden in Kapitel 2 der MNIST-Datensatz und seine Besonderheiten erläutert; anschließend werden in Kapitel 3 ein einfaches MLP und ein Basis-CNN als Vergleichsmodelle aufgebaut (vgl. Anhang 2/3). Darauf aufbauend wird ab Kapitel 3.2 ff. das CNN schrittweise erweitert, indem zentrale Bausteine (u. a. Aktivierungsfunktionen, Initialisierung, Regularisierung, Trainingsorganisation sowie Optimizer/Lernratensteuerung) systematisch variiert und ausgewertet werden. Am Ende folgt eine vergleichende Analyse, in der das optimierte CNN den beiden Basismodellen gegenübergestellt wird.

2 Datensatz

2.1 Überblick: Was ist MNIST

Der MNIST-Datensatz (Modified National Institute of Standards and Technology database) ist ein weit verbreiteter Standarddatensatz zur Klassifikation handschriftlich geschriebener Ziffern von 0 bis 9. Er besteht aus insgesamt 70.000 Graustufenbildern mit einer Auflösung von jeweils 28×28 Pixeln – also exakt 784 Pixel pro Bild. Die Daten sind aufgeteilt in 60.000 Trainings- und 10.000 Testbilder. Jedes Bild enthält genau eine Ziffer und ist mit einem Label versehen, das der dargestellten Zahl entspricht. Ziel ist es, ein Modell zu entwickeln, das allein auf Grundlage des Bildes die richtige Ziffer vorhersagen kann.

2.2 Besonderheiten von Bilddaten

Bilddaten unterscheiden sich grundlegend von klassischen tabellarischen Daten. Sie sind hochdimensional – ein einzelnes Bild enthält 784 Merkmalswerte – und weisen eine starke räumliche Struktur auf. Die Tatsache, dass MNIST aus Graustufenbildern besteht (Wertebereich 0–255) reduziert die Farbinformation und erleichtert somit das Training, da nur eine einzige Intensitätsdimension verarbeitet werden muss (Kantardzic, 2019, S. 12–13). Im Gegensatz zu klassischen „Summary-Statistics-Berechnungen“ wie Mittelwert oder Standardabweichung sind bei Bildern oft visuelle oder geometrische Muster hilfreich, um die Daten besser einordnen zu können (vgl. Abbildung 1).

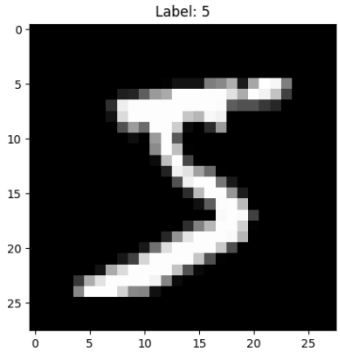

Abbildung 1, Abbild eines beispielhaften Datensatzes

2.3 Notwendige Vorbereitung des Datensatzes

Um neuronale Netze effektiv und richtig trainieren zu können, ist die Datenvorbereitung essenziell. Im Rahmen dieses Projekts wurden folgende Anpassungen vorgenommen:

- **Trainings-, Validierungs- und Testdaten-Split:** Von den insgesamt 60 000 Trainingsbildern wurden 90 % (54 000) für das eigentliche Training der Modelle genutzt. Die verbleibenden 10 % (6 000) bilden den Validierungsdatensatz, der nicht zum Lernen der Gewichte verwendet wird, sondern während des Trainings allein zur Überwachung dient – etwa für Early Stopping oder die dynamische Anpassung der Lernrate. Zusätzlich stehen die separaten 10 000 Testbilder zur Verfügung, die erst ganz am Ende eingesetzt werden, um die endgültige Modellleistung unabhängig und objektiv zu bewerten. Durch diese Dreiteilung wird eine saubere Trennung zwischen Lernen, Abstimmen und abschließender Evaluation gewährleistet.
- **Normalisierung**: Die Pixelwerte liegen ursprünglich im Bereich von 0 (schwarz) bis 255 (weiß). Wenn man diese Werte unverändert verwendet, kann es während des Trainings zu instabilen Gradienten und starken Gewichtsschwankungen kommen. Das erschwert die Konvergenz des Modells oder kann sogar dazu führen, dass es nicht lernt. Durch das Skalieren auf den Bereich [0, 1] – also die Division durch 255 – werden alle Eingabewerte vergleichbarer und liegen in einem Bereich, mit dem neuronale Netze effizienter und stabiler arbeiten können (vgl. Abbildung 2).

```
# Normalisieren auf [0,1] und in float32 umwandeln
x_train_raw = (x_train_raw / 255.0).astype("float32")
x_test_raw  = (x_test_raw  / 255.0).astype("float32")
```

Abbildung 2, Codebeispiel: Daten normalisieren

- **One-Hot-Encoding**: Die Zielwerte (Ziffern 0–9) sind kategoriale Klassen und müssen in eine Form gebracht werden, die das Modell korrekt interpretieren kann. Beim One-Hot-Encoding wird jede Klasse durch einen Vektor der Länge 10 dargestellt, wobei genau eine Position auf 1 gesetzt ist – diejenige, die zur jeweiligen Ziffer gehört – und alle anderen Positionen auf 0 (vgl. Abbildung 3, 4). Dieses Verfahren verhindert, dass das Modell die Klassen numerisch fehlinterpretiert, etwa indem es „9“ für ähnlicher zu „8“ als zu „1“ hält. One-Hot-Encoding ist insbesondere bei Verwendung der Softmax-Aktivierung in der Ausgabeschicht sinnvoll. Alternativ wäre auch die Verwendung von Ganzzahlen (Integer Labels) möglich – jedoch nur, wenn eine passende Verlustfunktion wie Sparse Categorical Crossentropy gewählt wird. In der Praxis ist One-Hot-Encoding der Standard, da es universell einsetzbar und kompatibel mit vielen Trainingsansätzen ist.

```
# One-Hot-Encoding
y_tr   = to_categorical(y_tr_int,  num_classes=10)
y_val  = to_categorical(y_val_int, num_classes=10)
y_test = to_categorical(y_test_raw, num_classes=10)
```

Abbildung 3, Codebeispiel: One-Hot-Encoding

```
Original Label (y_train[0]): 5
One-Hot-Encoded (y_train[0]): [0. 0. 0. 0. 0. 1. 0. 0. 0. 0.]
```

Abbildung 4, Ausgabebeispiel: One-Hot-Encoding

- **Formatierung**: Die Daten müssen so umgewandelt werden, dass sie dem erwarteten Eingabeformat des jeweiligen neuronalen Netzes entsprechen. Beispielsweise werden für Multilayer Perceptrons (MLP) flache Vektoren benötigt und für Convolutional Neural Networks (CNN) bildähnliche Strukturen mit Höhe, Breite und Kanal (genauere Erklärungen folgen in Anhang 2 und 3).

Was bei MNIST nicht berücksichtigt werden muss**:** Im Vergleich zu vielen realweltlichen Bilddatensätzen ist MNIST einfach gehalten. Die Bilder sind bereits zentriert, gleich groß (28×28), enthalten keine Farbinformationen (Graustufen) und die Klassen sind ausgewogen verteilt. Es gibt keine Metadaten, keine Hintergrundstörungen und keine fehlenden Werte. Dadurch entfällt weiterer Bedarf an Datenbereinigung, Resizing, Balancing oder komplexer Vorverarbeitung.

3 Aufbau der Modelle

Im folgenden Abschnitt wird die Architektur sowohl eines MLP als auch eines CNN systematisch optimiert. Die Analyse erfolgt schrittweise entlang der folgenden Themenbereiche:

- Phase 1: Grundlegender Architektur Kern
- Phase 2: Aktivierungsfunktion und Gewichtsinitialisierung
- Phase 3: Regularisierung
- Phase 4: Trainingsorganisation
- Phase 5: Optimizer & Lernraten (LR) – Steuerung

Zur Bewertung der unterschiedlichen Modellvarianten werden Metriken aus zwei Perspektiven herangezogen:

- Effektivität: Accuracy, F1-Score und Generalization Gap als Indikator für Overfitting.
- Effizienz: Parameteranzahl, Inferenzzeit pro Beispiel, gesamte Trainingszeit, beste Epoche sowie die bis dahin verstrichene Zeit (Time-to-Best).

3.1 Phase 1: Grundlegender Architektur Kern

Um die Ergebnisse fair und vergleichbar zu halten, werden folgende Trainingsparameter zunächst einheitlich festgelegt und nicht verändert. Zum Einsatz kommen der Adam-Optimizer (Lernrate 0,001) in Kombination mit categorical crossentropy als Loss-Funktion (Standard für Mehrklassenklassifikation mit Softmax-Ausgabe, daher nicht gesondert untersucht), eine Batchgröße von 64, maximal 20 Epochen sowie Early Stopping mit *patience=3* und *restore_best_weights=True*. Zur Reproduzierbarkeit wird der Seed auf 0 gesetzt. Die Initialisierung und Aktivierung der versteckten Schichten erfolgen mit Rectified Linear Unit (ReLU) und HeNormal, in der Ausgabeschicht wird Softmax mit GlorotUniform verwendet. Regularisierungsmethoden (z. B. Dropout oder Batch Normalization) werden in dieser frühen Phase noch nicht berücksichtigt, um die Effekte der Architektur klar isolieren zu können.

Den Beginn bildet das MLP, bei dem die Anzahl der Schichten sowie deren Neuronenanzahl systematisch variiert werden. Die Ergebnisse zeigen deutlich, dass vor allem die ersten Schritte bei der Erhöhung der Neuronenzahl den größten Einfluss auf die Modellgüte haben. Während Variante 1 (1×8 Neuronen) lediglich eine Accuracy von 92,4 % erreicht, steigert bereits Variante 2 (1×16 Neuronen) die Leistung auf 94,9 % – ein

Zuwachs von 2,5 %-Punkten bei vergleichsweise geringer Erhöhung der Parameterzahl. Weitere Vergrößerungen liefern jedoch abnehmenden Ertrag: So verbessern Variante 7 (64/32 Neuronen) und Variante 8 (2×64 Neuronen) die Genauigkeit nur noch auf rund 97,0–97,2 %, erfordern dafür aber mehr als 50 000 Parameter. Auch der Effekt zusätzlicher Schichten zeigt sich differenziert. Kleine Zwei-Schicht-Modelle wie Variante 3 (16/8 Neuronen**)** oder Variante 4 (2×16 Neuronen) bringen kaum Fortschritte gegenüber einer einzelnen Schicht. Erst größere Konfigurationen wie Variante 5 (32/16 Neuronen) oder besonders Variante 6 (2×32 Neuronen) können die Genauigkeit spürbar steigern. Eine dritte Schicht, wie in Variante 10 (64/32/16 Neuronen) oder Variante 11 (128/64/32 Neuronen)**,** erhöht die Accuracy zwar geringfügig auf maximal 97,4 %, benötigt dafür jedoch über 100 000 Parameter und ist damit deutlich weniger effizient. Als klarer Kompromiss aus Leistung und Aufwand erweist sich Variante 6 (2×32 Neuronen). Mit 96,2 % Accuracy & F1, nur ~26 500 Parametern und der kürzesten Trainingszeit (~4,9 s) bietet sie ausreichend Kapazität, um die Strukturen des Datensatzes zuverlässig zu erfassen, ohne dass die Modellkomplexität unnötig ansteigt. Größere Architekturen erzielen lediglich marginale Zugewinne, erkaufen diese jedoch mit erheblich höherem Rechenaufwand (vgl. Abbildung 5).

MLP	1. Variante	2. Variante	3. Variante	4. Variante	5. Variante	6. Variante	7. Variante	8. Variante	9. Variante	10. Variante	11. Variante
	1 verstecke Schicht mit 8 N.	1 verstecke Schicht mit 16 N.	2 versteckte Schichten mit 16 und 8 N.	2 versteckte Schichten mit je 16 N.	2 versteckte Schichten mit 32 und 16 N.	2 versteckte Schichten mit je 32 N.	2 versteckte Schichten mit 64 und 32 N.	2 versteckte Schichten mit je 64 N.	2 versteckte Schichten mit 128 und 64 N.	3 versteckte Schichten mit 64, 32 und 16 N.	3 versteckte Schichten mit 128, 64 und 32 N.
Accuracy	0.92350	0.94867	0.95183	0.94617	0.95767	0.96217	0.97017	0.97167	0.97433	0.96800	0.97417
F1	0.92270	0.94810	0.95143	0.94560	0.95725	0.96189	0.97002	0.97149	0.97431	0.96774	0.97405
Parameteranzahl	6.370	12.730	12.786	13.002	25.818	26.506	52.650	55.050	109.386	53.018	111.146
Trainingszeit in s	7.306	7.616	7.081	6.746	5.246	4.876	4.561	5.058	4.990	5.375	4.627

Abbildung 5, Wählen der Standardarchitekur für das MLP-Baseline-Model

Das CNN wird zunächst mit einfachen Standardeinstellungen aufgebaut (beispielsweise ohne Padding und mit Max-Pooling). Ausgangspunkt ist ein Block aus einer Convolutional-, Pooling-, Flatten- und Dense-Schicht, bei dem die Anzahl der Filter sowie Neuronen variiert werden. Anschließend wird die Netzarchitektur durch zusätzliche Schichten erweitert, um den Einfluss von Tiefe und Breite zu untersuchen. Im nächsten Schritt soll dann der Einfluss spezifischer Eigenschaften wie Kernelgrößen, Padding-Varianten oder Pooling-Methoden analysiert werden.

Beim Vergleich der CNN-Varianten zeigt sich, dass bereits kleine Architekturen eine hohe Genauigkeit erreichen. Variante 1 (8 Filter, 8 Neuronen) erzielt mit nur rund 11 000 Parametern 97,3 % Accuracy, während eine Verdopplung der Filter auf 16 (Variante 2) kaum Verbesserungen bringt, die Parameterzahl jedoch verdoppelt. Deutlich stärker wirkt

hingegen die Erhöhung der Neuronen in der Dense-Schicht: Variante 3 (8 Filter, 16 Neuronen) steigert die Genauigkeit auf 97,9 %, ohne nennenswert mehr Parameter zu erfordern. Auffällig ist damit, dass die Anpassung der Dense-Schicht einen größeren Einfluss auf die Modellgüte hat als die Vergrößerung der Convolutional-Schicht. Dies ist plausibel, da der MNIST-Datensatz überwiegend aus einfachen Kanten- und Strukturen besteht und viele Filter (32 oder 64) für die relativ geringen Merkmalsunterschiede überdimensioniert sind. Bereits bei Variante 4 (16 Filter, 16 Neuronen) zeigt sich eine Sättigung: die Accuracy steigt nur noch minimal, während die Parameterzahl auf über 43 000 wächst. Noch größere Modelle wie Variante 5 (32 Filter, 16 Neuronen) oder Variante 6 (16 Filter, 32 Neuronen) erreichen zwar 98,2–98,4 %, benötigen dafür jedoch schon rund 87 000 Parameter. Ab Variante 7 bis 10 (≥ 32 Filter und ≥ 32 Neuronen) bleibt die Genauigkeit mit 98,3–98,5 % nahezu konstant, während die Modellgröße exponentiell anwächst und bis fast 700 000 Parameter erreicht. Der beste Kompromiss liegt bei den mittleren Varianten, insbesondere Variante 3 (8 Filter, 16 Neuronen) und Variante 4 (16 Filter, 16 Neuronen), die mit rund 98 % Genauigkeit und moderater Modellgröße ein ausgewogenes Verhältnis zwischen Leistung und Aufwand bieten (vgl. Abbildung 6)

CNN	1. Variante	2. Variante	3. Variante	4. Variante	5. Variante	6. Variante	7. Variante	8. Variante	9. Variante	10. Variante
	jeweils 1 Conv2D- (8 F.), Pooling und Dense Schicht (8 N.)	jeweils 1 Conv2D- (16 F.), Pooling und Dense Schicht (8 N.)	jeweils 1 Conv2D- (8 F.), Pooling und Dense Schicht (16 N.)	jeweils 1 Conv2D- (16 F.), Pooling und Dense Schicht (16 N.)	jeweils 1 Conv2D- (32 F.), Pooling und Dense Schicht (16 N.)	jeweils 1 Conv2D- (16 F.), Pooling und Dense Schicht (32 N.)	jeweils 1 Conv2D- (32 F.), Pooling und Dense Schicht (32 N.)	jeweils 1 Conv2D- (64 F.), Pooling und Dense Schicht (32 N.)	jeweils 1 Conv2D- (32 F.), Pooling und Dense Schicht (64 N.)	jeweils 1 Conv2D- (64 F.), Pooling und Dense Schicht (64 N.)
Accuracy	0.97283	0.97317	0.97850	0.97917	0.98217	0.98367	0.98317	0.98167	0.98350	0.98483
F1	0.97282	0.97292	0.97843	0.97913	0.98214	0.98365	0.98315	0.98165	0.98350	0.98485
Parameteranzahl	10.994	21.890	21.898	43.610	87.034	87.050	173.738	347.114	347.146	693.578
Trainingszeit in s	27.921	39.567	28.856	25.500	40.459	34.912	27.885	36.734	21.194	37.199

Abbildung 6, Erhöhung der Neuronen Anzahl unter Verwendung einer Convolutional- und Dense-Schicht

Im direkten Vergleich zur Ein-Schicht-Architektur aus Abbildung 6 zeigt die Einführung einer zweiten Conv2D-Schicht sofort Wirkung. Schon die kleinste Zwei-Conv-Variante, Variante 1 (8,16 Filter, Dense 8), erreicht 98,35 % Accuracy bei nur 4 546 Parametern – ein Niveau, das in Abbildung 5 erst große Ein-Conv-Modelle wie (16 Filter, 32 Neuronen**)** erreichen (98,37 %, 87 050 Parameter). Dies lässt sich dadurch begründen, dass die zweite Conv-Schicht auf den durch Pooling reduzierten Feature Maps aufsetzt. Während die erste Schicht hauptsächlich einfache Muster wie Kanten oder Striche erkennt, kann die zweite Schicht diese Merkmale zu komplexeren Strukturen wie Kurven oder Ziffernteilen zusammenfassen. Dadurch entstehen aussagekräftigere Merkmalskarten, sodass bereits sehr kleine Zwei-Conv-Modelle die Leistung großer Ein-Conv-Netze erzielen, dabei aber ein Vielfaches an Parametern einsparen. Insgesamt stellt Variante 4 (16 Filter, 16 Neuronen) eine sinnvolle Wahl dar. Sie kombiniert eine sehr hohe Genauigkeit von

98,63 % mit einer moderaten Parameteranzahl von nur 17 756 und kurzer Trainingszeit. Modelle mit mehr Filtern und Neuronen (Varianten 5 – 7) steigern die Accuracy kaum noch (max. +0,2 %), führen jedoch zu einer erheblichen Erhöhung der Parameteranzahl und damit des Rechenaufwands (vgl. Abbildung 7).

CNN	1. Variante	2. Variante	3. Variante	4. Variante	5. Variante	6. Variante	7. Variante
	2 Conv2D-(8, 16 F.), Pooling und 1 Dense Schicht (8 N.)	2 Conv2D-(8, 16 F.), Pooling und 1 Dense Schicht (16 N.)	2 Conv2D-(16, 32 F.), Pooling und 1 Dense Schicht (8 N.)	2 Conv2D-(16, 32 F.), Pooling und 1 Dense Schicht (16 N.)	2 Conv2D-(32, 64 F.), Pooling und 1 Dense Schicht (8 N.)	2 Conv2D-(32, 64 F.), Pooling und 1 Dense Schicht (16 N.)	2 Conv2D-(32, 64 F.), Pooling und 1 Dense Schicht (32 N.)
Accuracy	0.98350	0.98400	0.97633	0.98633	0.98467	0.98750	0.98767
F1	0.98345	0.98391	0.97616	0.98632	0.98466	0.98754	0.98770
Parameteranzahl	4.546	7.834	11.298	17.786	31.714	44.602	70.378
Trainingszeit in s	39.773	33.336	73.643	34.132	85.183	53.191	54.190

Abbildung 7, Einfluss zweier Convolutional- und Pooling-Schichten

Wird anstelle einer zweiten Convolutional-Schicht eine zusätzliche Dense-Schicht eingeführt, zeigt sich auch in dieser Versuchsreihe nur ein sehr begrenzter Nutzen. Die Accuracy bleibt über alle Varianten hinweg im Bereich von 97,6–98,4 % und liegt damit kaum über den Ergebnissen einfacher Architekturen mit nur einer Dense-Schicht. Besonders deutlich wird dies im Vergleich: Variante 1 (8 Filter, Dense 16/8) erreicht 97,7 %, während selbst die komplexeste Konfiguration Variante 6 (32 Filter, Dense 32/16) lediglich 98,4 % erzielt. Der Leistungsgewinn beträgt damit weniger als ein Prozentpunkt, obwohl die Parameterzahl von rund 22 000 auf über 174 000 ansteigt. Auch die Trainingszeiten liefern kein klares Argument für tiefere Dense-Strukturen. Selbst die höchste erzielte Genauigkeit mit Variante 6 (98,6 %) bleibt hinter den Zwei-Conv-Architekturen zurück (vgl. Abbildung 8): So erreicht beispielsweise Variante 4 aus Abbildung 7 mit nur 17 786 Parametern bereits eine höhere Genauigkeit von 98,7 %. Insgesamt zeigt sich somit, dass zusätzliche Dense-Schichten beim MNIST-Datensatz keinen Mehrwert bringen. Der geringe Zuwachs an Genauigkeit steht in keinem Verhältnis zum starken Anstieg der Parameteranzahl und führt nicht zu einer besseren Generalisierung. Statt die Abstraktionsfähigkeit des Netzes zu verbessern, erhöhen sie lediglich die Komplexität und machen die Modelle ineffizienter.

CNN	1. Variante	2. Variante	3. Variante	4. Variante	5. Variante	6. Variante
	1 Conv2D-(8 F.), Pooling und 2 Dense Schichten (16, 8 N.)	1 Conv2D-(8 F.), Pooling und 2 Dense Schichten (32, 16 N.)	1 Conv2D-(16 F.), Pooling und 2 Dense Schichten (16, 8 N.)	1 Conv2D-(16 F.), Pooling und 2 Dense Schichten (32, 16 N.)	1 Conv2D-(32 F.), Pooling und 2 Dense Schichten (16, 8 N.)	1 Conv2D-(32 F.), Pooling und 2 Dense Schichteb (32, 16 N.)
Accuracy	0.97717	0.97783	0.97617	0.97933	0.97900	0.98400
F1	0.97713	0.97777	0.97613	0.97937	0.97897	0.98398
Parameteranzahl	21.954	44.074	43.666	87.418	87.090	174.106
Trainingszeit in s	26.556	15.979	23.752	19.042	33.489	28.032

Abbildung 8, Einfluss zweiter Dense-Schicht

Im nächsten Schritt wird geprüft, ob die Kombination aus zwei Conv2D- und Pooling-Schichten mit zwei Dense-Schichten einen Mehrwert gegenüber dem bislang besten Modell (Variante 4 aus Abbildung 7: 98,63 % Accuracy, 17 786 Parameter, 34 s Trainingszeit) bringt. Schon Variante 1 erreicht mit 98,6 % nahezu die gleiche Genauigkeit wie das Referenzmodell, benötigt dafür jedoch deutlich mehr Ressourcen. Auch stärkere Ausprägungen wie Variante 2 (98,9 %) oder Variante 4 (98,95 %) liefern nur minimale Zugewinne, während die Parameterzahl und die Trainingszeit jeweils stark ansteigen. Teilweise bleiben die Modelle sogar leicht hinter der Referenz zurück, wie etwa Variante 3 (98,5 %) (vgl. Abbildung 9). Insgesamt wird deutlich, dass zwei Dense-Schichten keinen echten Vorteil bringen: Die Genauigkeit verbessert sich nur marginal, während Komplexität und Rechenaufwand erheblich zunehmen. Damit bestätigt sich, dass beim MNIST-Datensatz eine zusätzliche Dense-Schicht keinen substanziellen Mehrwert bietet. Die klaren, kontrastreichen Strukturen lassen sich bereits mit einer einzigen kompakten Dense-Schicht zuverlässig klassifizieren, während zusätzliche Schichten lediglich zu unnötiger Modellkomplexität führen.

CNN	1. Variante	2. Variante	3. Variante	4. Variante
	2 Conv2D-(16, 32 F.), Pooling und 2 Dense Schichten (16, 8 N.)	2 Conv2D-(16, 32 F.), Pooling und 2 Dense Schichten (32, 16 N.)	2 Conv2D-(32, 64 F.), Pooling und 2 Dense Schichten (16, 8 N.)	2 Conv2D-(32, 64 F.), Pooling und 2 Dense Schichten (32, 16 N.)
Accuracy	0.98550	0.98883	0.98450	0.98950
F1	0.98554	0.98881	0.98455	0.98950
Parameteranzahl	17.842	31.130	44.658	70.746
Trainingszeit in s	56.220	64.785	60.466	62.123

Abbildung 9, Einfluss zweier Con2D-, Pooling- und Dense-Schichten

Für MNIST bleibt daher die Architektur mit zwei Conv2D-Schichten (16 und 32 Filter) sowie einer Dense-Schicht mit 16 Neuronen der beste Kompromiss: Sie erreicht eine sehr hohe Accuracy (98,63 %) bei moderater Parameteranzahl (17 786) und kurzen Laufzeiten und stellt damit die optimale Baseline dar (vgl. Abbildung 7). Ausgehend von diesem

Modell, werden jetzt nacheinander einzelne Parameter (Kernelgröße, Padding, Pooling) verändert, um die optimale Kern Architektur für das CNN-Netz zu definieren:

Kernelgröße:

Die Tests zur Kernelgröße zeigen, dass die Wahl der Filtergröße einen deutlichen Einfluss auf Genauigkeit und Effizienz hat. Während 3×3-Kernel bereits sehr gute Ergebnisse erzielen (Accuracy 98,63 %**,** 17 786 Parameter, Inferenzzeit 0,021 ms), können 5×5-Kernel die Genauigkeit leicht steigern (98,93 %), gehen jedoch mit höherer Parameteranzahl (21 626) und längerer Inferenzzeit (0,027 ms) einher. Zu große Kernel (7×7) sind für den kleinen MNIST-Datensatz ineffizient, da sie Details verwischen und die Genauigkeit sogar verschlechtern (98,57 %, 28 154 Parameter). Besonders interessant ist die Kombination 5×5 in der ersten und 3×3 in der zweiten Convolutional-Schicht: Hier werden zunächst grobe Strukturen erfasst und anschließend durch kleinere Filter verfeinert. Diese Variante erzielt 98,87 % Genauigkeit bei gleichzeitig moderater Modellgröße (18 042 Parameter) und einer Trainingszeit von 45 s und stellt damit den optimalen Kompromiss für den Datensatz dar (vgl. Abbildung 10).

CNN	1. Variante	2. Variante	3. Variante	4. Variante
2 Conv2D- (16, 32 F.), Pooling und 1 Dense Schicht (16 N.)	Kernel 2 x (3x3)	Kernel 2 x (5x5)	Kernel 2 x (7x7)	Kernel (5x5) und (3x3)
Accuracy	0.98633	0.98933	0.98567	0.98867
F1	0.98632	0.98935	0.98562	0.98867
Parameteranzahl	17786	21626	28.154	18.042
Trainingszeit in s	32.397	60.610	39.535	45.446
Inferenzzeit pro Beispiel in ms	0.02056	0.02715	0.02200	0.02329

Abbildung 10, Einfluss der Kernelgrößen

Padding:

Der Vergleich zeigt, dass das Modell ohne Padding („valid“) die bessere Wahl für den MNIST-Datensatz ist. Ohne Padding erreicht es eine etwas höhere Genauigkeit (98,87 % vs. 98,73 %), benötigt deutlich weniger Parameter (18 042 vs. 30 330) und ist sowohl im Training (48,9 s vs. 57,5 s) als auch in der Inferenz schneller (0,027 ms vs. 0,034 ms) (vgl. Abbildung 11). Der Grund liegt darin, dass die Ziffern im MNIST-Datensatz zentriert dargestellt sind und die Randbereiche hauptsächlich aus Pixeln bestehen, die keinen Mehrwert zur Identifizierung der Zahlen beitragen. Padding fügt dort künstlich zusätzliche Randinformationen hinzu, die keinen Mehrwert liefern, sondern lediglich die Modellkomplexität und Rechenzeit erhöhen. Somit ist für diesen Datensatz Padding nicht notwendig und sogar leicht nachteilig.

CNN	1. Variante	2. Variante
2 Conv2D- (16, 32 F.), Pooling und 1 Dense Schicht (16 N.)	ohne Padding (standard)	mit Padding
Accuracy	0.98867	0.98733
F1	0.98867	0.98726
Parameteranzahl	18.042	30.330
Trainingszeit in s	48.865	57.500
Inferenzzeit pro Beispiel in ms	0.02739	0.03383

Abbildung 11, Einfluss von Padding

Pooling:

Der Vergleich zeigt deutliche Unterschiede zwischen den Pooling-Varianten und den eingesetzten Kernelgrößen. Max Pooling mit einem 2×2-Kernel erzielt die beste Balance: Mit 98,87 % Test-Accuracy liefert es die stabilsten Ergebnisse. Ein größerer Kernel (3×3) reduziert die Genauigkeit leicht auf 98,65 %, da die stärkere Verdichtung filigrane Muster weniger präzise abbildet – dafür sinkt die Parameteranzahl deutlich (7290 statt 18 042), und auch die Trainingszeit ist mit 54,7 s niedriger. Average Pooling erreicht ebenfalls solide Ergebnisse (98,83 % bei 2×2, 98,48 % bei 3×3), fällt aber konsistent etwas schwächer aus, da das Mitteln Kontraste glättet und damit markante Kanten und Striche abschwächt. Global Pooling schneidet deutlich schlechter ab: Mit nur 96,88 % (Max) bzw. 95,33 % (Average) Accuracy geht zu viel räumliche Information verloren, da jede Feature-Map auf einen einzigen Wert reduziert wird. Für den MNIST-Datensatz ist Max Pooling (2×2) die sinnvollste Wahl, da es entscheidende Bilddetails erhält, gleichzeitig eine moderate Modellgröße (18 042 Parameter) beibehält und mit einer Trainingszeit von 51,1 s eine gute Effizienz zeigt (vgl. Abbildung 12).

CNN	1. Variante	2. Variante	3. Variante	4. Variante	3. Variante	4. Variante
2 Conv2D- (16, 32 F.), Pooling und 1 Dense Schicht (16 N.)	Max Pooling (2x2)	Max Pooling (3x3)	Average Pooling (2x2)	Average Pooling (3x3)	Global Max Pooling	Global Average Pooling
Accuracy	0.98867	0.98650	0.98833	0.98483	0.96883	0.95333
F1	0.98867	0.98640	0.98823	0.98486	0.96853	0.95275
Parameteranzahl	18.042	7290	18.042	7.290	5.754	5.754
Trainingszeit in s	51.083	54.721	55.627	53.790	176.714	172.719
Inferenzzeit pro Beispiel in ms	0.02531	0.02207	0.02730	0.02310	0.04360	0.05102

Abbildung 12, Einfluss der verschiedenen Pooling Varianten mit Kernelgrößen

Reihenfolge vom Pooling sowie Stride-2 Convs:

In diesem Test wurde untersucht, welchen Einfluss die Platzierung von Pooling-Operationen sowie der Einsatz von Stride-2 Convolutions auf die Modellleistung hat. Bei einer Stride-2-Faltung bewegt sich der Filter nicht pixelweise (stride=1,1 als Standard), sondern in Schritten von zwei Pixeln (stide= 2,2) über das Bild. Dadurch wird die räumliche

Auflösung der Feature-Maps bereits während der Faltung reduziert. Im Gegensatz zu Pooling, das eine feste, nicht lernbare Verdichtung darstellt, erfolgt die Reduktion hier „gelernt“: Die Filtergewichte bestimmen gleichzeitig, welche Muster verstärkt und welche verworfen werden. Stride-2 kombiniert somit Downsampling und Merkmalsextraktion in einem einzigen Schritt.

Pooling nach jeder Conv2D-Schicht (Variante 1) liefert mit 98,87 % Accuracy die besten Resultate. Durch die zweifache Reduktion der Feature-Maps bleibt der Input der Dense-Schicht klein (18 042 Parameter), was das Modell effizient und robust macht. Wird hingegen nur nach der zweiten Conv2D-Schicht gepoolt (Variante 2), verschlechtert sich die Balance deutlich: Die Genauigkeit sinkt leicht (98,53 %**),** während Parameterzahl (67 194) und Trainingszeit (59,7 s) stark steigen – verursacht durch große Feature-Maps und einen entsprechend aufgeblähten Flatten-Vektor. Auch der Einsatz von Stride-2 führt zu einer geringeren Genauigkeit (Variante 3: 98,55 %; Variante 4: 98,42 %) und erhöht gleichzeitig die Modellkomplexität (56–67k Parameter) (vgl. Abbildung 13). Zwar entfallen explizite Pooling-Layer, doch Stride-2 erfasst Verschiebungen im Bild weniger zuverlässig als MaxPooling. Dadurch können kleine Lageänderungen von Ziffern schlechter abgefangen werden, und bei einem Datensatz wie MNIST gehen feine Strukturen verloren – was sich unmittelbar in einer schwächeren Performance niederschlägt. Für MNIST bestätigt sich folglich, dass frühes und regelmäßiges Pooling die effizienteste und genaueste Lösung darstellt. Stride-2 ist eine interessante Alternative, bietet in diesem Szenario aber keinen Vorteil und führt eher zu Genauigkeitsverlusten.

CNN	1. Variante	2. Variante	3. Variante	4. Variante
2 Conv2D- (16, 32 F.), Pooling und 1 Dense Schicht (16 N.)	nach jeder Conv2D-Schicht Pooling	nur nach der zweiten Conv2D-Schicht Pooling	Stride-2 in der ersten Conv2D-Schicht	Stride-2 in der zweiten Conv2D-Schicht
Accuracy	0.98867	0.98533	0.98550	0.98417
F1	0.98867	0.98532	0.98540	0.98409
Parameteranzahl	18.042	67.194	56.442	67.194
Trainingszeit in s	50.652	59.746	14.243	32.462
Inferenzzeit pro Beispiel in ms	0.02405	0.04195	0.01693	0.02446

Abbildung 13, Einfluss der Reihenfolge vom Pooling sowie Stride-2 Convs

Abschließend wurden die beiden Grundmodelle direkt miteinander verglichen. Dabei zeigt sich, dass das CNN – wie aufgrund seiner Spezialisierung auf Bilddaten zu erwarten – das MLP deutlich übertrifft. Dennoch erreicht das MLP mit einer Accuracy von 96,2 % ein starkes Ergebnis, liegt damit aber rund 2,7 Prozentpunkte hinter dem CNN (vgl. Abbildung 14). In allen betrachteten Kennzahlen schneidet das CNN besser ab, einzig die

Trainingszeit fällt deutlich höher aus. Aufgrund seiner klaren Überlegenheit und seiner prinzipiellen Eignung für Bildverarbeitung wird lediglich das CNN in den folgenden Phasen verwendet und schrittweise weiter optimiert.

Architektur Kern	1. Variante	2. Variante
	MLP-Architektur Model	CNN-Architektur Model
Accuracy	0.96217	0.98867
F1	0.96189	0.98867
Parameteranzahl	26.506	18.042
Trainingszeit in s	4.940	41.520
Inferenzzeit pro Beispiel in ms	0.00654	0.02168

Abbildung 14, Vergleich des favorisierten MLP- und CNN-Netzes nach Phase 1

3.2 Phase 2: Aktivierungsfunktion und Gewichtsinitialisierung

In Phase zwei wird zuerst die Wahl der Aktivierungsfunktion untersucht, danach die der Gewichtsinitialisierung, während alle anderen Trainingsparameter unverändert bleiben. Aktivierungsfunktionen haben in neuronalen Netzen die zentrale Aufgabe, die lineare Berechnung eines Neurons – also die gewichtete Summe seiner Eingaben – in eine nichtlineare Ausgabe zu überführen. Ohne diese zusätzliche Transformation könnte ein Netz, selbst wenn es viele Schichten umfasst, nur lineare Zusammenhänge abbilden. Mehrere lineare Funktionen hintereinander ergeben nämlich wieder nur eine lineare Funktion. Damit wäre das Modell in seiner Ausdruckskraft stark eingeschränkt und könnte komplexe Muster in den Daten nicht erfassen. Durch die Aktivierungsfunktion wird dieser lineare Zusammenhang „verbogen“: Je nach Form der Funktion werden Eingaben abgeschnitten, abgeschwächt oder in einem glatten Übergang weitergegeben. Dadurch entstehen nichtlineare Entscheidungsgrenzen, mit denen das Netz auch komplizierte Strukturen und hochdimensionale Muster erkennen kann. Aktivierungsfunktionen steuern außerdem, wie Gradienten im Training weitergegeben werden, und haben damit einen direkten Einfluss auf Stabilität und Effizienz des Lernprozesses.

Für die Untersuchung wurden vier gängige Aktivierungsfunktionen berücksichtigt. Die Rectified Linear Unit (ReLU) gibt positive Eingaben unverändert weiter und setzt negative Werte auf Null. Sie ist recheneffizient, verhindert Sättigung im positiven Bereich und gilt als Standard in Convolutional Neural Networks. Die LeakyReLU erweitert diesen Ansatz, indem auch negative Eingaben mit einem kleinen Faktor weitergegeben werden, wodurch „tote Neuronen“ vermieden werden können. Die Gaussian Error Linear Unit (GELU) nutzt eine glatte, probabilistische Übergangsfunktion und erlaubt eine weichere

Aktivierung, die insbesondere in sehr tiefen Architekturen Vorteile zeigen kann. Schließlich sorgt die Scaled Exponential Linear Unit (SELU) für eine selbstnormalisierende Dynamik, die Aktivierungen während des Trainings automatisch stabilisiert. Dieser Effekt tritt allerdings vor allem in tieferen Netzen ohne Batch Normalization auf.

Die Ergebnisse der Experimente zeigen, dass sich alle Varianten mit Ausnahme von SELU in Bezug auf Accuracy und F1-Score nur minimal unterscheiden. ReLU (Variante 1) erreicht mit 98,98 % Accuracy und einem F1-Score von 0,9898 den besten Wert und weist eine kürzere Trainingszeit auf als GELU. LeakyReLU (Variante 2) und GELU (Variante 3) erreichen ebenfalls vergleichbare Werte, liefern jedoch keinen klaren Vorteil: LeakyReLU verbessert weder Genauigkeit noch Stabilität, GELU verlängert das Training spürbar. SELU schneidet mit 98,7 % Accuracy und 0,9868 F1 klar schwächer ab, was darauf zurückzuführen ist, dass seine selbstnormalisierenden Eigenschaften in der hier verwendeten flachen CNN-Struktur nicht greifen (vgl. Abbildung 15). Insgesamt lässt sich ableiten, dass ReLU die überzeugendste Wahl darstellt. Sie liefert nahezu identische Modellgüte wie die komplexeren Varianten, bleibt dabei aber effizienter in Parameterzahl, Trainingsdauer und Inferenzgeschwindigkeit. Damit wird ReLU als Standard-Aktivierungsfunktion für die folgenden Phasen festgelegt.

Aktivierungsfunktion	1. Variante	2. Variante	3. Variante	4. Variante
	ReLU	LeakyReLU	GELU	SELU
Accuracy	0.98983	0.98967	0.98967	0.98683
F1	0.98980	0.98972	0.98958	0.98678
Parameteranzahl	18.042	18.042	18.042	18.042
Trainingszeit in s	51.871	44.544	82.698	44.737
Inferenzzeit pro Beispiel in ms	0.02095	0.02027	0.04850	0.02260

Abbildung 15, Auswirkungen verschiedener Aktivierungsfunktionen

Gewichtsinitialisierungen legen die Startwerte der Modellparameter vor dem eigentlichen Training fest und haben damit einen entscheidenden Einfluss auf die Lernfähigkeit eines neuronalen Netzes. Werden die Gewichte ungünstig gewählt, kann dies zu Problemen wie stagnierendem Training, explodierenden oder verschwindenden Gradienten führen. Eine sinnvolle Initialisierung sorgt dafür, dass die Verteilungen der Aktivierungen und Gradienten in allen Schichten stabil bleiben und sich das Netz effizient trainieren lässt. Damit wird der Grundstein für eine stabile und schnelle Konvergenz gelegt, ohne dass zusätzliche Trainingszeit oder komplexe Korrekturmechanismen notwendig sind.

Für die Untersuchung wurden drei gängige Initialisierungsmethoden berücksichtigt. Die HeNormal-Initialisierung zieht die Gewichte aus einer normalverteilten Zufallsvariable,

deren Varianz speziell auf ReLU-Aktivierungen zugeschnitten ist (Varianzskalierung nach fan_in). Dadurch werden Aktivierungen und Gradienten in tiefen Netzen stabil gehalten, was diese Methode zum Standard für ReLU-Netze macht. Die HeUniform-Initialisierung basiert auf demselben Prinzip, verwendet jedoch eine gleichverteilte Zufallsvariable im passenden Intervall. Sie bietet ebenfalls eine gute Anpassung an ReLU und kann im Einzelfall leicht stabilere Konvergenz liefern. Die Glorot- oder Xavier-Uniform-Initialisierung ist eine allgemeinere Methode, die die Varianz der Gewichte so wählt, dass Eingaben und Ausgaben einer Schicht im Gleichgewicht bleiben (Varianzskalierung nach fan_avg). Ursprünglich für Sigmoid- und Tanh-Aktivierungen entwickelt, kann sie auch mit ReLU eingesetzt werden, ist dort jedoch nicht immer optimal.

Die Ergebnisse der Experimente zeigen, dass alle drei Methoden zu sehr ähnlichen Resultaten führen. HeNormal (Variante 1) erreicht mit 98,98 % Accuracy und einem F1-Score von 0,9898 eine solide Modellgüte bei der kürzesten Trainings- (51,3 s) und Inferenzzeit (0,0203 ms). HeUniform (Variante 4) erzielt mit 99,07 % Accuracy und 0,9907 F1 die besten Werte in Bezug auf Güte, benötigt jedoch etwas mehr Trainingszeit (55,1 s) und ist in der Inferenz geringfügig langsamer. Glorot/Xavier Uniform (Variante 5) liegt mit 99,00 % Accuracy und 0,9900 F1 zwischen beiden Varianten, ohne jedoch signifikante Vorteile in Effizienz oder Stabilität zu zeigen (vgl. Abbildung 16). Insgesamt lässt sich zusammenfassen, dass HeNormal und HeUniform in ihrer Güte praktisch gleichauf liegen. Während HeUniform die leicht höheren Genauigkeitswerte liefert, bietet HeNormal Vorteile bei Effizienz und Laufzeit. HeNormal gilt allerdings in der Literatur als die Standardinitialisierung für ReLU-Netze weshalb Variante 1 als favorisiertes Modell für die folgenden Phasen festgelegt wird.

Gewichtsinitialisierung	1. Variante	2. Variante	3. Variante
	HeNormal	HeUniform	Glorot/Xavier Uniform
Accuracy	0.98983	0.99067	0.99000
F1	0.98980	0.99066	0.99002
Parameteranzahl	18.042	18.042	18.042
Trainingszeit in s	51.261	55.074	53.947
Inferenzzeit pro Beispiel in ms	0.02030	0.02121	0.02139

Abbildung 16, Auswirkungen verschiedener Gewichtsinitialisierungen

3.3 Phase 3: Regularisierung

Für die dritte Phase des Projekts wurden die Trainings- und Validierungseinstellungen angepasst, um den Einfluss von Regularisierungsmethoden angemessen bewerten zu

können. Die maximale Epochenzahl wurde auf 60 erhöht und die Early-Stopping-Patience auf 5 gesetzt. Regularisierungsmethoden wie Dropout oder L2-Regularisierung verlangsamen häufig die Konvergenz. Durch mehr Spielraum und etwas höhere Geduld wird ein vorzeitiges Abbrechen vermieden, ohne die Laufzeit unnötig zu verlängern, da Early Stopping ohnehin vorzeitig stoppt. Zusätzlich wurde die Callback-Funktion *ReduceLROnPlateau* integriert. Diese halbiert die Lernrate (0,001) automatisch, sobald sich der Validierungsverlust über mehrere Epochen nicht verbessert. Damit können regulärisierte Modelle auch in Plateaus die letzten Verbesserungen erreichen, ohne dass die Lernrate manuell angepasst werden muss. Für Modellvarianten mit Batch Normalization wird eine größere Batchgröße (128 statt 64) verwendet. Dadurch werden die Batch-Statistiken stabiler berechnet und die Effizienz pro Epoche erhöht. Für alle anderen Runs bleibt die Batchgröße unverändert bei 64. Die jeweilige Einstellung wird pro Variante dokumentiert.

Die Auswahl der getesteten Regularisierungsmethoden basiert auf ihrem unterschiedlichen Wirkmechanismus im Lernprozess: Dropout schaltet während des Trainings zufällig Neuronen ab und verhindert damit, dass das Netzwerk zu stark von einzelnen Aktivierungen abhängig wird. L2-Regularisierung bestraft große Gewichtswerte und erzwingt kleinere, gleichmäßigere Gewichte, wodurch die Modellkomplexität reduziert und die Generalisierung verbessert wird. Batch Normalization stabilisiert die Verteilungen innerhalb der Schichten und sorgt für einen gleichmäßigeren Gradientenfluss, was vor allem bei tieferen Architekturen einen positiven Effekt auf die Trainingsdynamik haben kann. Durch die Kombination dieser Ansätze lassen sich verschiedene Aspekte von Overfitting adressieren.

Wird anstelle einer unregulierten Basisarchitektur Dropout eingesetzt, zeigt sich, dass ein Wert von 0.2 die Generalisierung verbessert, ohne die Genauigkeit stark zu beeinträchtigen (Variante 2: 98,9 %, Gap –0.013). Bei stärkerer Regularisierung mit 0.5 sinkt die Accuracy jedoch auf 98,8 % und der Gap fällt deutlich negativ aus (Variante 3: –0.215), was auf Underfitting hinweist. Die L2-Regularisierung erweist sich als wirksam: Bereits mit 1e-4 steigt die Accuracy leicht auf 99,1 % (Variante 4), mit 5e-4 wird sogar der beste Wert unter den Einzellösungen erzielt (Variante 5: 99,2 %). Allerdings verlängern sich die Trainingszeiten dabei deutlich, insbesondere bei stärkerer Gewichtung (285 s). Batch Normalization zeigt im Vergleich keinen klaren Vorteil. Trotz stabilerer Aktivierungen bleibt die Accuracy auf 98,8 % begrenzt (Variante 6) und liegt damit nicht über den Ergebnissen ohne Regularisierung (98,9 %, Variante 1). Dies erklärt sich durch die

Einfachheit von MNIST und die geringe Netztiefe. Besonders interessant sind die Kombinationen: Dropout 0.2 in Verbindung mit L2 (5e-4) erreicht mit 99,3 % die höchste Genauigkeit (Variante 8), während Dropout 0.2 mit BatchNorm eine ähnlich hohe Accuracy von 99,2 % liefert (Variante 9), jedoch mit nahezu perfekter Stabilität (Gap – 0.002). Zu starke Kombinationen wie Dropout 0.5 mit L2 (Variante 10/11) zeigen dagegen deutliche Nachteile und führen zu Underfitting. Insgesamt lässt sich festhalten, dass Dropout 0.2 + L2 (5e-4) (Variante 8) die höchste Genauigkeit erzielt, während Dropout 0.2 + BatchNorm (Variante 9) die stabilste Generalisierung bietet. Beide Varianten stellen damit die favorisierten Lösungen dar (vgl. Abbildungen 17 und 18).

Regularisierung	1. Variante	2. Variante	3. Variante	4. Variante	5. Variante	6. Variante	7. Variante
	keine Regularisierung	Dropout 0.2 (nach Dense-Schicht)	Dropout 0.5 (nach Dense-Schicht)	L2 (1e-4) (nach Dense-Schicht)	L2 (5e-4) (nach Dense-Schicht)	BatchNorm (vor jeder Aktivierung)	Dropout 0.2 + L2 (1e-4)
Accuracy	0.98983	0.98933	0.98833	0.99067	0.99200	0.98817	0.98950
F1	0.98980	0.98933	0.98831	0.99062	0.99197	0.98818	0.98946
Generalisierungslücke	0.00870	-0.01311	-0.21546	0.00887	0.00800	0.01183	-0.01589
Zeit bis zur besten Epoche	46.526	51.221	57.249	112.161	285.077	115.845	62.839
Beste Epoche	14	15	18	34	60	19	15

Abbildung 17, Auswirkungen der Regularisierungsvarianten Teil 1

Regularisierung	8. Variante	9. Variante	10. Variante	11. Variante	12. Variante	13. Variante	14. Variante
	Dropout 0.2 + L2 (5e-4)	Dropout 0.2 + BatchNorm	Dropout 0.5 + L2 (1e-4)	Dropout 0.5 + L2 (5e-4)	Dropout 0.5 + BatchNorm	L2 (1e-4) + BatchNorm	L2 (5e-4) + BatchNorm
Accuracy	0.99283	0.99183	0.98917	0.99017	0.99167	0.98850	0.98800
F1	0.99280	0.99183	0.98915	0.99013	0.99168	0.98850	0.98798
Generalisierungslücke	-0.01104	-0.00196	-0.13957	-0.11715	-0.11126	0.01150	0.01200
Zeit bis zur besten Epoche	150.031	151.196	45.239	79.689	107.980	113.740	156.886
Beste Epoche	45	24	15	25	18	20	27

Abbildung 18, Auswirkungen der Regularisierungsvarianten Teil 2

Da die Ergebnisse der einzelnen Läufe nur geringe Unterschiede zeigen, werden die beiden favorisierten Varianten sowie die Ausgangsvariante ohne Regularisierung (Variante 1) jeweils fünfmal erneut trainiert. Durch die anschließende Betrachtung der Mittelwerte entsteht ein verlässlicher Vergleich, der aufzeigt, welche Variante sich tatsächlich als die leistungsstärkste erweist und welchen Mehrwert sie im Vergleich zum Basismodell liefert.

Die Auswertung der fünf Wiederholungen bestätigt zunächst, dass beide regulärisierten Varianten einen leichten, aber konsistenten Vorteil gegenüber der unregulierten Ausgangsarchitektur (Variante 1) bieten. So liegt die mittlere Accuracy des Basismodells bei 98,97 %, während sowohl Variante 8 (Dropout 0.2 + L2 5e-4) als auch Variante 9

(Dropout 0.2 + BatchNorm) mit 99,1 % ein etwas höheres Niveau erreichen. Auch der F1-Score zeigt ein entsprechendes Bild. Entscheidend für die Bewertung ist jedoch die Generalisierung: Während das unregulierte Modell mit einem positiven Gap (+0.008) erwartungsgemäß leicht zum Overfitting neigt, weist Variante 8 ein stark negatives Gap (–0.017) auf, was auf eine Überkompensation durch die Regularisierung hindeutet. Variante 9 liefert dagegen mit –0.003 einen nahezu idealen Wert und erreicht damit die stabilste Balance zwischen Trainings- und Validierungsleistung. Bei der Trainingsdauer zeigt sich, dass Variante 8 im Mittel etwas schneller zur besten Epoche findet (35 Epochen, 125 s) als Variante 9 (26 Epochen, 135 s) (vgl. Abbildung 19). Angesichts der fast identischen Genauigkeit und F1-Werte überwiegt jedoch die bessere Generalisierung von Variante 9. Damit bestätigt sich, dass Regularisierung grundsätzlich gerechtfertigt ist: Dropout 0.2 + BatchNorm (Variante 9) erweist sich dabei als die Variante, die hohe Genauigkeit mit besonders stabiler Generalisierung verbindet und wird folglich als favorisierte Lösung ausgewählt, welche die Grundlage für die weitere Optimierung in den folgenden Phasen bildet.

Regularisierung	1. Variante	8. Variante	9. Variante
Mitellwert von fünf Durchläufen	keine Regular-isierung	Dropout 0.2 + L2 (5e-4)	Dropout 0.2 + BatchNorm
Accuracy	0.98973	0.99097	0.99097
F1	0.98971	0.99095	0.99096
Generalisierungslücke	0.00843	-0.01686	-0.00330
Zeit zur besten Epoche in s	52.9	125.3	135.4
Beste Epoche	19	35	26

Abbildung 19, Wiederholter Test mit den favorisierten Regularisierungsvarianten

3.4 Phase 4: Trainingsorganisation

In Phase vier wurde ausschließlich die Batch-Size variiert, während alle anderen Trainingsparameter wie Optimizer, Lernrate und Early Stopping unverändert blieben. Die Batch-Size ist für die Trainingsorganisation besonders relevant, da sie bestimmt, wie viele Trainingsbeispiele in einem Schritt gemeinsam verarbeitet werden. Kleine Batches führen zu mehr Aktualisierungen pro Epoche, wodurch die Gradienten verrauschter sind. Dies kann die Generalisierung begünstigen, verlangsamt aber das Training. Große Batches reduzieren die Zahl der Aktualisierungen pro Epoche, liefern in der Regel stabilere Gradienten und erhöhen die Recheneffizienz, bergen jedoch die Gefahr einer geringfügig schlechteren Generalisierung. Ziel dieser Phase ist, die Auswirkungen gängiger Batch-

Größen (32, 64, 128, 265) auf Genauigkeit, Generalisierung und Effizienz des bisher favorisierten Modells systematisch zu untersuchen.

Die Ergebnisse zeigen zunächst, dass die Genauigkeit und der F1-Score über alle Varianten hinweg nahezu identisch sind (Unterschied < 0.001). Dies weist darauf hin, dass die Wahl der Batch-Size in diesem Szenario die reine Modellgüte kaum beeinflusst. Unterschiede zeigen sich jedoch in den sekundären Metriken. Bei der Generalisierungslücke fällt auf, dass die kleinere Batch-Size von 32 mit –0.005 den größten Unterschied zwischen Trainings- und Validierungsgenauigkeit aufweist (Variante 1). Dies deutet auf eine geringfügig schlechtere Generalisierung hin. Die mittleren bis großen Batches (64–256) liefern mit Werten um –0.003 stabilere Ergebnisse (Variante 2-4). Die Effizienz des Trainings zeigt klare Vorteile für größere Batches: Während Batch 64 mit durchschnittlich 146 Sekunden die längste Zeit bis zur besten Epoche benötigt, erreichen die Varianten 3 und 4 mit Batch 128 (122 s) und 256 (116 s) deutlich schneller ihr Optimum. Zwar benötigen größere Batches mehr Epochen (von 19 bei Batch 32 bis 30 bei Batch 256), die Gesamtdauer sinkt jedoch, da die einzelnen Epochen schneller berechnet werden. Besonders deutlich wird der Vorteil größerer Batches bei der Inferenzzeit: Während Batch 64 hier mit 0.049 ms pro Beispiel die langsamste Variante (2) darstellt, liefern Batch 128 und 256 mit 0.027 ms die effizientesten Werte. Batch 32 liegt mit 0.032 ms dazwischen, kann sich aber ebenfalls nicht gegen die größeren Varianten durchsetzen (vgl. Abbildung 25).

Insgesamt lässt sich ableiten, dass die beiden Varianten mit Batch 128 und 256 die besten Ergebnisse erzielen, da sie Genauigkeit, Generalisierung und Effizienz am überzeugendsten kombinieren. Zwischen diesen beiden fällt die Wahl jedoch auf Variante 4 (Batch 256): Sie erreicht mit 115,6 Sekunden die kürzeste Zeit bis zur besten Epoche, bietet identische Inferenzgeschwindigkeit wie Batch 128 und weist ein ebenso stabiles Generalisierungsverhalten auf. Der minimale Unterschied in der Genauigkeit von nur 0.0003 ist nicht aussagekräftig. Damit liefert Batch 256 praktisch dieselbe Modellgüte wie Batch

128, jedoch mit höherer Trainingseffizienz (vgl. Abbildung 20). Folglich wurde Batch 256 als Standardkonfiguration für die folgende Phase ausgewählt.

Batch-Size Größe	1. Variante	2. Variante	3. Variante	4. Variante
Mitellwert von fünf Durchläufen	32	64	128	256
Accuracy	0.99137	0.99093	0.99097	0.99067
F1	0.99138	0.99094	0.99096	0.99067
Generalisierungslücke	-0.00502	-0.00304	-0.00330	-0.00312
Zeit zur besten Epoche in s	128.2	145.7	121.9	115.6
Beste Epoche	19	23	26	30
Inferenzzeit pro Beispiel in ms	0.032	0.049	0.027	0.027

Abbildung 20, Auswirkungen der Batch-Size Größe

3.5 Phase 5: Optimizer & Lernraten (LR) – Steuerung

In Phase fünf wird die Wahl des Optimizers sowie die Steuerung der Lernrate untersucht. Optimizer und Lernratensteuerung bestimmen maßgeblich, wie ein neuronales Netz während des Trainings seine Gewichte anpasst. Der Optimizer legt dabei fest, wie die Gradienten des Fehlers in Aktualisierungen der Gewichte übersetzt werden, während die Lernrate steuert, wie groß diese Anpassungsschritte ausfallen. Eine zu hohe Lernrate kann das Training instabil machen und dazu führen, dass das Netz nicht konvergiert, während eine zu niedrige Lernrate die Konvergenz stark verlangsamt oder in lokale Minima führt. Um dieses Problem abzufedern, wurde neben einer festen Lernrate auch ein dynamisches Verfahren getestet. Der Scheduler ReduceLROnPlateau überwacht den Validierungsverlust und halbiert die Lernrate automatisch, sobald sich dieser über mehrere Epochen hinweg nicht mehr verbessert. Auf diese Weise werden schnelle Lernfortschritte zu Beginn mit einer feineren Anpassung in späteren Phasen kombiniert.

Für die Untersuchung wurden zwei Optimizer jeweils mit fester Lernrate sowie in Kombination mit ReduceLROnPlateau getestet. Der stochastische Gradientenabstieg (SGD) ist der klassische Optimizer. Dabei werden die Gewichte nach jedem Mini-Batch in Richtung des Gradienten der Verlustfunktion angepasst. Alle Parameter teilen sich eine feste Lernrate, die für das gesamte Training gilt. Das macht SGD recheneffizient und gut kontrollierbar, kann aber auch dazu führen, dass das Training langsam verläuft oder in Plateaus steckenbleibt, da alle Gewichte unabhängig von ihrer Bedeutung im Modell gleich stark angepasst werden. Der Adam-Optimizer (Adaptive Moment Estimation) erweitert dieses Prinzip durch eine adaptive Steuerung. Er merkt sich nicht nur den aktuellen Gradienten, sondern auch den Mittelwert und die Streuung vergangener Gradienten für jedes Gewicht separat. Daraus ergibt sich für jedes Gewicht eine eigene effektive Lernrate:

Parameter mit stabilen, gleichmäßigen Gradienten werden stärker angepasst, während Parameter mit stark schwankenden Gradienten vorsichtiger aktualisiert werden. Dadurch kann Adam schneller konvergieren, robuster auf unterschiedliche Skalen im Modell reagieren und kommt mit weniger manueller Feinabstimmung der Lernrate aus.

Die Ergebnisse der Experimente zeigen, dass Adam und SGD mit fester Lernrate in Bezug auf Accuracy und F1 nahezu gleichauf liegen (jeweils rund 99,0 %). Unterschiede treten vor allem in der Trainingseffizienz zutage: Adam erreicht das Optimum bereits nach 14 Epochen und 58,7 Sekunden, während SGD erst nach 30 Epochen und 134,3 Sekunden konvergiert. Mit ReduceLROnPlateau wird der Vorteil von Adam noch deutlicher: Die Kombination erzielt die besten Werte mit 99,12 % Accuracy und 0,9911 F1 und weist mit –0.003 einen nahezu idealen Generalization Gap auf. Zwar verlängert sich die Trainingsdauer (113,7 Sekunden), doch der Zugewinn an Genauigkeit und Generalisierung überwiegt diesen Nachteil (vgl. Abbildung 21). SGD mit ReduceLROnPlateau bringt dagegen keinen erkennbaren Vorteil und bleibt sowohl bei Genauigkeit als auch bei Trainingseffizienz hinter den Adam-Varianten zurück. Insgesamt lässt sich ableiten, dass Adam die überzeugendste Basis darstellt: Er ist schneller, effizienter und ebenso genau wie SGD. Der Einsatz von ReduceLROnPlateau verstärkt diesen Vorteil zusätzlich, da er die Genauigkeit und Stabilität des Modells sichtbar verbessert. Damit wird Adam in Kombination mit ReduceLROnPlateau (Variante 3) als favorisierte Einstellung gewählt.

Optimizer & Lernrate	1. Variante	2. Variante	3. Variante	4. Variante
Mitellwert von fünf Durchläufen	Adam	SGD	Adam mit ReduceLR OnPlateau	SDG mit ReduceLR OnPlateau
Accuracy	0.99050	0.99017	0.99117	0.98833
F1	0.99054	0.99014	0.99115	0.98827
Zeit zur besten Epoche in s	58.714	134.337	113.693	123.842
Generalisierungslücke	-0.00822	-0.00735	-0.00300	-0.00878
Beste Epoche	14	30	26	27
Inferenzzeit pro Beispiel in ms	0.03005	0.02919	0.02956	0.03146

Abbildung 21, Auswirkungen des Optimizers und der Lernraten-Steuerung

3.6 Analyse der festgelegten Netze

Ausgehend vom in Phase eins entwickelten CNN-Basisnetz wurde das Modell Schritt für Schritt erweitert (vgl. Kapitel 3.2–3.5) und gezielt optimiert. Dieses optimierte CNN-Netz soll nun erstmals den beiden ursprünglichen Basismodellen – dem CNN-Basisnetz sowie einem einfachen MLP – gegenübergestellt werden, um die Wirkung der

Optimierung beurteilen zu können. Dafür werden alle drei Modelle auf den bislang ungenutzten Testdaten evaluiert und systematisch analysiert.

Das optimierte CNN umfasst zwei Convolutional-Schichten mit 16 und 32 Feature Maps (Kernelgrößen 5×5 bzw. 3×3), jeweils kombiniert mit Batch Normalization und ReLU-Aktivierung. Nach beiden Convolutional-Schritten erfolgt MaxPooling (2×2). Anschließend folgt eine Flatten-Schicht sowie eine Dense-Schicht mit 16 Neuronen, ergänzt durch Dropout (0,2) zur Regularisierung. Die Ausgabeschicht besteht aus 10 Neuronen mit Softmax-Aktivierung und GlorotUniform-Initialisierung (vgl. Abbildung 22). Trainiert wird das Netz mit dem Optimizer Adam (Lernrate 1e-3), categorical_crossentropy als Loss-Funktion, einer Batch Size von 256 sowie den Callbacks Early Stopping (Patience = 5) und ReduceLROnPlateau.

```python
def build_cnn_regularized():

    model = Sequential()

    # Eingabe-Schicht
    model.add(Input(shape=(28, 28, 1)))

    # CNN-Blöcke mit BatchNormalization und Dropout
    model.add(Conv2D(16, (5,5), padding='valid', kernel_initializer=HeNormal(), use_bias=False, activation=None))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(MaxPooling2D((2,2)))

    model.add(Conv2D(32, (3,3), padding='valid', kernel_initializer=HeNormal(), use_bias=False, activation=None))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(MaxPooling2D((2,2)))

    model.add(Flatten())

    model.add(Dense(16, kernel_initializer=HeNormal(), activation=None, use_bias=False))
    model.add(BatchNormalization())
    model.add(Activation("relu"))
    model.add(Dropout(0.2))

    # Ausgabe-Schicht
    model.add(Dense(10, activation='softmax', kernel_initializer=GlorotUniform()))

    return model
```

Abbildung 22, Optimiertes CNN-Netz: Code

Im Vergleich dazu sind die beiden Basismodelle deutlich einfacher gehalten:

Das CNN-Basisnetz gleicht dem optimierten Modell im Aufbau der Schichten, verzichtet jedoch vollständig auf Regularisierungstechniken (kein Dropout, keine Batch Normalization) sowie auf eine adaptive Lernratensteuerung (vgl. Abbildung 23). Zudem arbeitet es mit einer Standard-Batch Size von 64.

```python
def build_cnn_basic():

    model = Sequential()

    # Eingabe-Schicht
    model.add(Input(shape=(28, 28, 1)))

    # CNN-Blöcke
    model.add(Conv2D(16, (5, 5), activation='relu', padding='valid', kernel_initializer=HeNormal()))
    model.add(MaxPooling2D((2, 2)))

    model.add(Conv2D(32, (3, 3), activation='relu', padding='valid', kernel_initializer=HeNormal()))
    model.add(MaxPooling2D((2, 2)))

    model.add(Flatten())

    model.add(Dense(16, activation='relu', kernel_initializer=HeNormal()))

    # Ausgabe-Schicht
    model.add(Dense(10, activation='softmax', kernel_initializer=GlorotUniform()))

    return model
```

Abbildung 23, Basis CNN-Netz: Code

Das MLP-Basisnetz besteht ausschließlich aus zwei verdeckten Dense-Schichten mit je 32 Neuronen, HeNormal-Initialisierung und ReLU-Aktivierung (vgl. Abbildung 24).

```python
def build_mlp_basic():

    model = Sequential()

    # Eingabe-Schicht
    model.add(Input(shape=(28 * 28,)))

    # Versteckte Schichten (manuell anpassbar)
    model.add(Dense(32, activation='relu', kernel_initializer=HeNormal()))
    model.add(Dense(32, activation='relu', kernel_initializer=HeNormal()))

    # Ausgabe-Schicht
    model.add(Dense(10, activation='softmax', kernel_initializer=GlorotUniform()))

    return model
```

Abbildung 24, Basis MLP-Netz: Code

Um eine faire Vergleichbarkeit sicherzustellen, werden beide Basismodelle für diese Analyse ebenfalls mit 60 Epochen und einer Patience von 5 trainiert.

Alle Modelle wurden mit 60 Epochen und einer Patience von 5 trainiert, um die Vergleichbarkeit sicherzustellen. In Abbildung 25 zeigt sich, dass die CNN-Modelle das MLP in der Accuracy klar übertreffen (96,7 % vs. 98,6 % bzw. 98,9 % Accuracy). Das optimierte CNN generalisiert am besten (Gap –0,003), benötigt jedoch die längste Zeit bis zur besten Epoche (121 s). MLP und Basis-CNN erreichen ihr Optimum deutlich schneller.

Finaler Test auf Testdaten	1. Variante	2. Variante	3. Variante
	MLP Basis	CNN Basis	CNN Optimiert
Accuracy	0.96670	0.98580	0.98950
F1	0.96635	0.98573	0.98944
Zeit zur besten Epoche in s	3.625	26.813	121.333
Generalisierungslücke	0.01359	0.00557	-0.00300
Beste Epoche	7	8	26
Inferenzzeit pro Beispiel in ms	0.00577	0.01823	0.02196
Parameteranzahl	26506	18042	18234

Abbildung 25, Vergleich Testdaten (MLP Basis, CNN Basis und CNN Optimiert)

Die Unterschiede in der Accuracy spiegeln sich in den Confusion-Matrizen wider: Sie erlauben eine detaillierte Betrachtung der Fehlklassifikationen. Von den 10 000 Testbildern werden beim MLP-Basismodell rund 333 falsch zugeordnet, was der Accuracy von 96,7 % entspricht. Die Klassengenauigkeit schwankt hier stärker zwischen 94 % (Ziffern 5 und 4) und 98 % (Ziffern 1, 7 und 9). Besonders häufig treten Verwechslungen zwischen 5 ↔ 3**,** 5 ↔ 8 sowie 9 ↔ 4 auf. Diese Fehler deuten darauf hin, dass das MLP Schwierigkeiten mit Ziffern hat, die sich in ihrer Form teilweise ähneln oder „offen/geschlossen" Varianten darstellen (vgl. Abbildung 26). Beim Basis-CNN sinkt die Zahl der Fehlklassifikationen deutlich auf etwa 140 Bilder. Die Klassengenauigkeit liegt nun fast durchgehend bei 99 %. Die Verwechslungen sind seltener, aber nicht völlig verschwunden. So werden vereinzelt noch 2 ↔ 8 oder 3 ↔ 5 falsch zugeordnet, während andere Problemfälle des MLP praktisch eliminiert sind (z. B. 9 ↔ 4). Das zeigt, dass das CNN deutlich robuster ist und bestimmte systematische Fehler des MLP gezielt behebt (vgl. Abbildung 27). Das optimierte CNN erreicht schließlich die beste Performance mit rund 105 Fehlklassifikationen. Fast alle Klassen erzielen hier eine Präzision von 99 %, nur die Ziffern 3, 5 und 8 liegen knapp darunter. Die verbleibenden Fehler konzentrieren sich auf wenige, hartnäckige Paare – etwa 3 ↔ 5 und in geringerem Maße 8 ↔ 3. Auffällig ist, dass mit zunehmender Modellkomplexität die Zahl der Fehler nicht nur sinkt, sondern sich auch die Breite der Fehlermuster reduziert: Während das MLP noch viele verschiedene Ziffernpaare verwechselt, beschränken sich die Fehler beim optimierten CNN auf wenige, klar erkennbare Fälle (vgl. Abbildung 28).

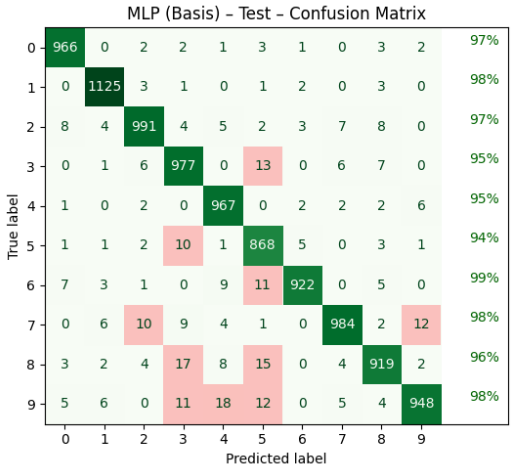

Abbildung 26, Confusion Matrix: MLP Basis

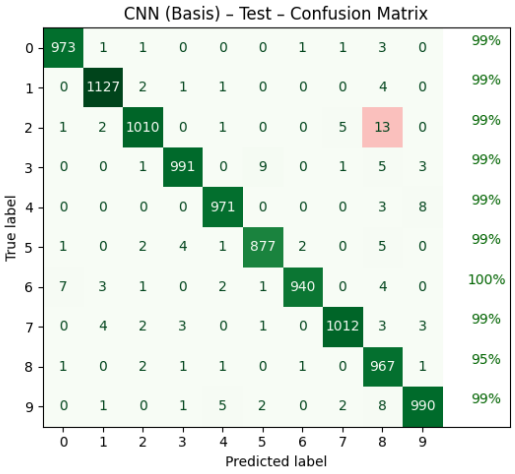

Abbildung 27, Confusion Matrix: CNN Basis

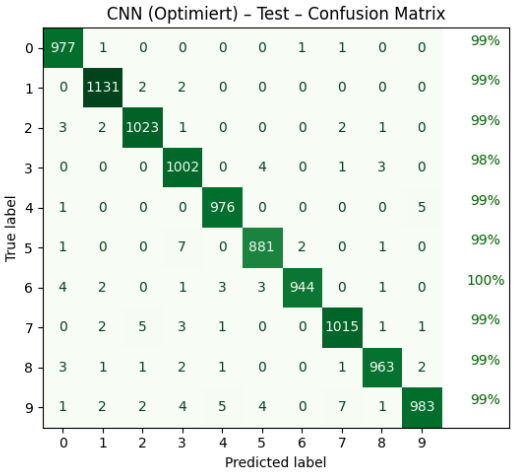

Abbildung 28, Confusion Matrix: CNN Optimiert

Dieses Erkenntnisse visualisiert die Heatmap ebenfalls: Beim MLP verteilt sich die Fehlerrate über viele Klassenpaare und erreicht in einzelnen Feldern Werte von über 30 %. Beim Basis-CNN konzentrieren sich die Verwechslungen auf wenige Cluster (z. B. 2 ↔ 8). Beim optimierten CNN verbleiben nur noch einzelne, schwach ausgeprägte Felder, was die in der Tabelle gezeigte hohe Genauigkeit und F1-Werte bestätigt (vgl. Abbildung 29-31).

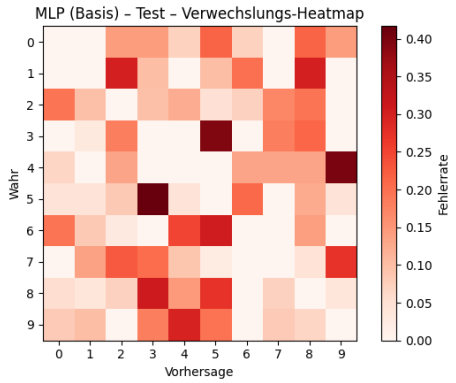

Abbildung 29, Heatmap: MLP Basis

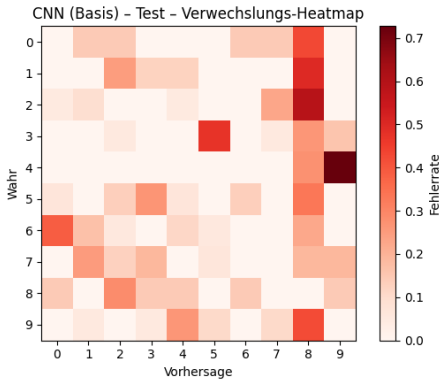

Abbildung 30, Heatmap: CNN Basis

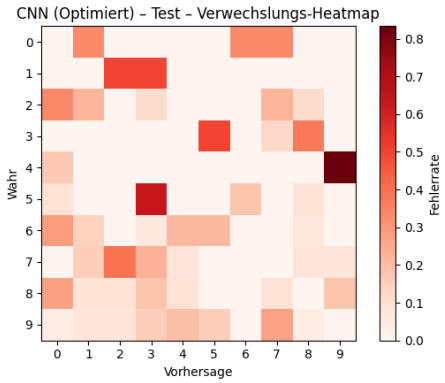

Abbildung 31, Heatmap: CNN Optimiert

Insgesamt zeigt sich: Die CNN-Modelle erzielen fast durchgehend 99 % Präzision pro Klasse, während das MLP in einzelnen Klassen deutlich abfällt. Mit steigender Modellqualität sinkt nicht nur die Gesamtfehlerzahl, sondern auch die Varianz zwischen den Klassen – ein Hinweis auf eine stabilere und ausgewogenere Klassifikationsleistung.

Die Lernkurven verdeutlichen die Unterschiede im Trainingsverhalten der Modelle. Beim MLP steigt die Trainingsgenauigkeit rasch an und erreicht über 98 %, während die Validierungsgenauigkeit bereits früh stagniert und bei etwa 96 % bleibt. Die Generalisierungslücke vergrößert sich kontinuierlich und beträgt in der besten Epoche rund +0,014, was die in Abbildung 30 angegebene Überanpassung bestätigt (vgl. Abbildung 32/33).

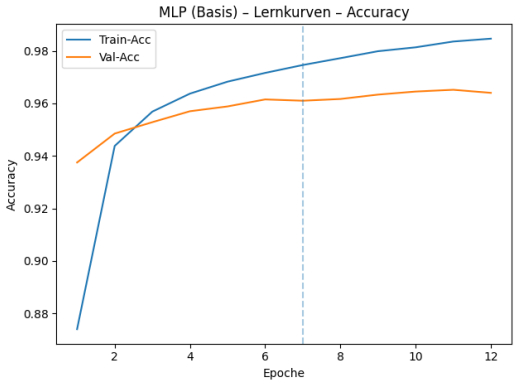

Abbildung 32, Accuracy: MLP Basis

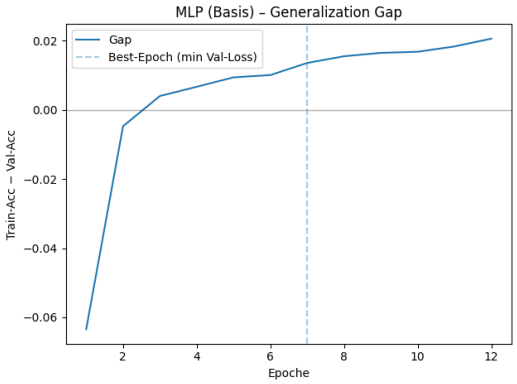

Abbildung 33, Generalisierungslücke: CNN Basis

Das Basis-CNN zeigt ein deutlich stabileres Verhalten. Train- und Val-Accuracy verlaufen fast parallel und erreichen Werte knapp unter 99 %. Der Gap bleibt über die gesamte Trainingsdauer gering (+0,006), was auf eine gute Balance zwischen Anpassung und Generalisierung hinweist (vgl Abbildung 34/35).

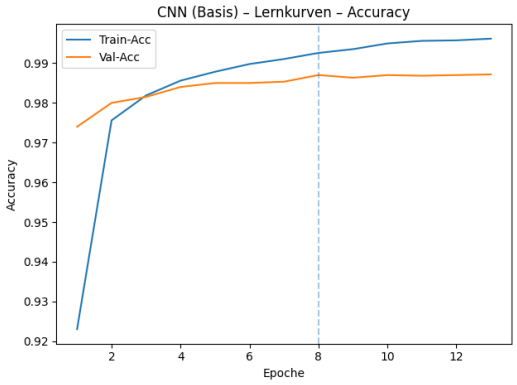

Abbildung 34, Accuracy: CNN Basis

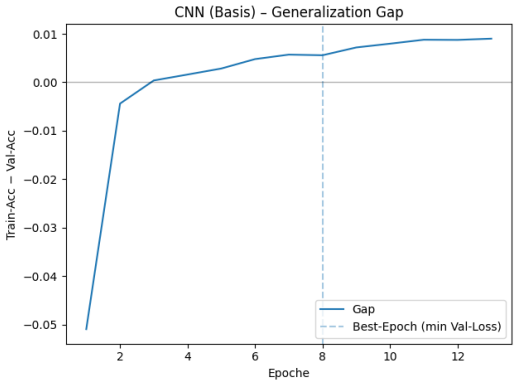

Abbildung 35, Generalisierungslücke: CNN Optimiert

Beim optimierten CNN ist schließlich eine besonders robuste Generalisierung zu beobachten: Die Validierungsgenauigkeit liegt über weite Teile des Trainings sogar leicht über der Trainingsgenauigkeit. Entsprechend ergibt sich ein minimal negativer Gap von –0,003, was darauf hinweist, dass das Modell durch Regularisierung und Lernratenanpassung besonders gut auf unbekannte Daten generalisiert (vgl. Abbildung 36/37).

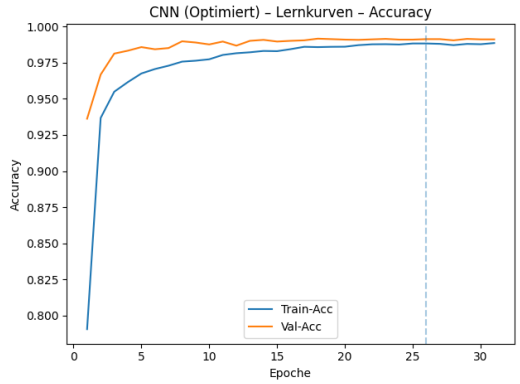

Abbildung 36, Accuracy: CNN Optimiert

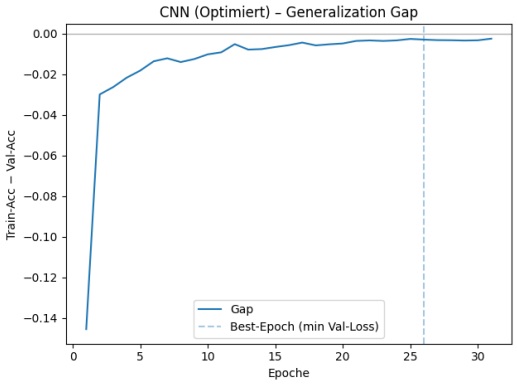

Abbildung 37, Generalisierungslücke: CNN Optimiert

Zusammenfassend zeigen die Kurven, dass das MLP zwar schnell konvergiert, jedoch stärker zum Overfitting neigt. Die CNN-Architekturen – insbesondere das optimierte Modell – erreichen eine deutlich höhere und zugleich stabilere Leistung, was bereits in den Confusion-Matrizen als größere Robustheit sichtbar wurde. Neben der reinen Genauigkeit stellt sich jedoch auch die Frage, wie zuverlässig die Modelle ihre Vorhersagewahrscheinlichkeiten einschätzen**.** Genau hier setzen die Reliability-Diagramme an: Sie zeigen, inwieweit die vorhergesagte Konfidenz mit der tatsächlichen Trefferrate übereinstimmt. Liegt ein Punkt auf der Diagonalen, dann stimmt die Sicherheit der Vorhersage exakt mit der realen Genauigkeit überein. Abweichungen davon zeigen Über- oder Unterkonfidenz an. Der Expected Calibration Error (ECE) fasst diese Abweichungen in einer Zahl zusammen – je kleiner, desto besser die Kalibrierung.

Alle drei Modelle sind sehr gut kalibriert, was angesichts des vergleichsweise einfachen MNIST-Datensatzes auch zu erwarten ist (0,005; 0,004 und 0,002). Ein Großteil der Testbeispiele wird mit extrem hoher Konfidenz (>0,9) klassifiziert – typisch für MNIST, da die meisten Ziffern sehr klar unterscheidbar sind. Entscheidend ist, dass diese hohe Sicherheit auch berechtigt ist: In allen drei Modellen entspricht die Trefferrate in diesem Bereich nahezu 100 %. Beim optimierten CNN konzentriert sich ein Großteil der Beispiele in diesem Bereich (Konfidenz und Trefferrate = 1), die Kurve liegt nahezu auf der Diagonalen und weist lediglich eine leichte Überkonfidenz ohne spürbare Abweichungen auf. Das Basis-CNN folgt demselben Muster einer „perfekten Ecke“, jedoch weniger ausgeprägt. Im mittleren bis hohen Konfidenzbereichen treten kleine Wellen durch teils stärkerer Überkonfidenz auf, wodurch der Verlauf etwas unruhiger wirkt. Das MLP liegt sehr nah an der Idealgeraden und schwankt nur minimal um sie herum. Im Vergleich zu den CNNs ist der Anteil perfekt sicherer Vorhersagen (Konfidenz = 1, Trefferrate = 1) jedoch geringer, was den etwas höheren ECE-Wert erklärt, obwohl die Kurve insgesamt

sehr „sauber“ erscheint. Zusammengefasst liefern alle Modelle verlässliche Wahrscheinlichkeiten, wobei das optimierte CNN die dichteste Ansammlung perfekter Vorhersagen mit einer sehr guten, diagonalen Kalibrierung verbindet (vgl. Abbildungen 38-40).

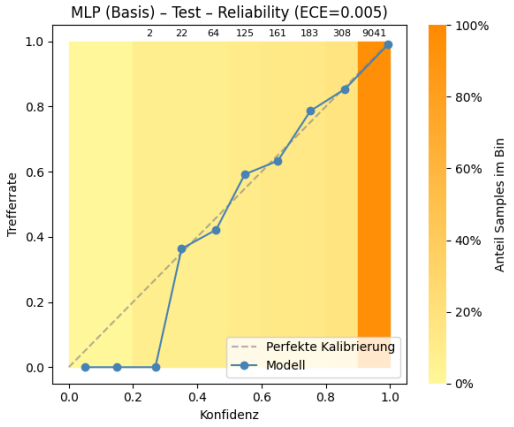

Abbildung 38, Reliability: MLP Basis

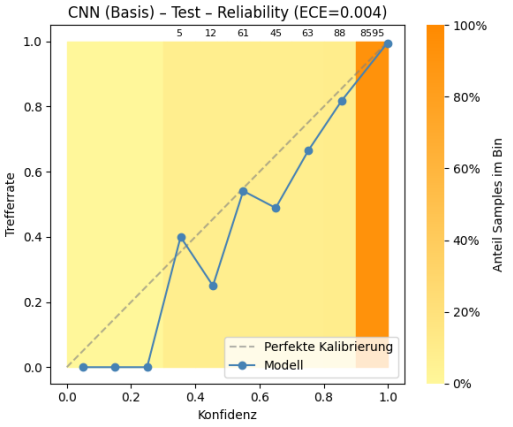

Abbildung 39, Reliability: CNN Basis

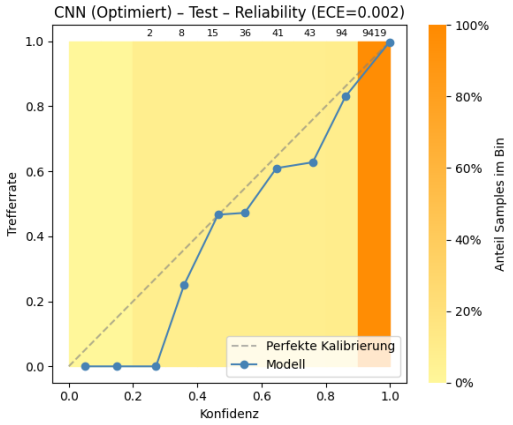

Abbildung 40, Reliability: Optimiertes CNN

Neben Genauigkeit und Kalibrierung ist auch die Robustheit von Bedeutung – also wie stabil ein Modell bei gestörten Eingabedaten bleibt. Dazu wurden die Testbilder schrittweise mit gaußschem Rauschen unterschiedlicher Stärke (Standardabweichung σ) überlagert und anschließend die Accuracy gemessen. Sinkt die Genauigkeit mit zunehmendem Rauschen nur langsam, deutet das auf eine hohe Robustheit gegenüber Bildstörungen hin (vgl. Abbildung 41).

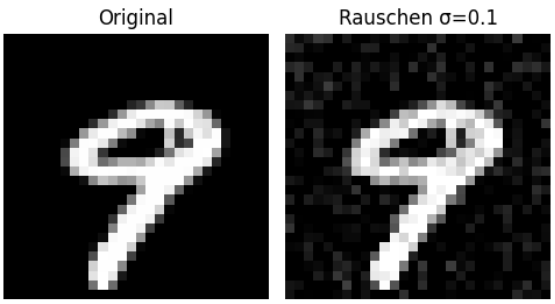

Abbildung 41, Beispielhaftes Rauschen mit Rauschstärke 0.1

Die Ergebnisse zeigen deutliche Unterschiede zwischen den Architekturen. Das MLP reagiert am empfindlichsten: Schon bei σ = 0,1 sinkt die Accuracy auf unter 94 %, und bei σ = 0,15 fällt sie auf nur noch rund 86 %. Damit bricht die Leistung stark ein, sobald die Eingabedaten gestört werden (vgl. Abbildung 42).

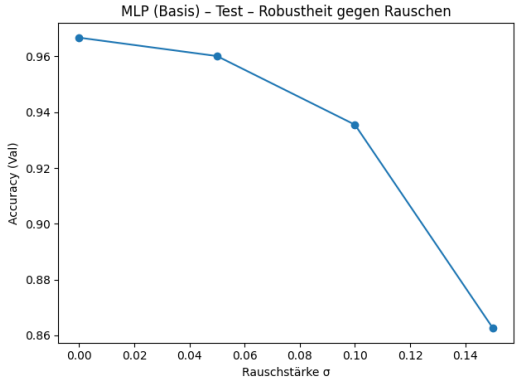

Abbildung 42, Rauschen: MLP Basis

Das Basis-CNN bleibt deutlich stabiler – bei leichtem Rauschen sind praktisch keine Verluste sichtbar, erst ab σ = 0,1 geht die Accuracy unter 98 %, bei σ = 0,15 liegt sie noch bei rund 97,8 %. Am robustesten zeigt sich das optimierte CNN: Es steigert die Accuracy bei schwachem Rauschen sogar minimal, bleibt bis σ = 0,1 über 98,5 % und unterschreitet selbst bei σ = 0,15 die Marke von 98 % nur knapp (vgl. Abbildung 43/44).

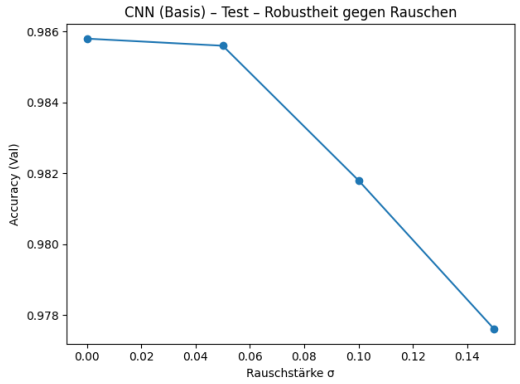

Abbildung 43, Rauschen: CNN Basis

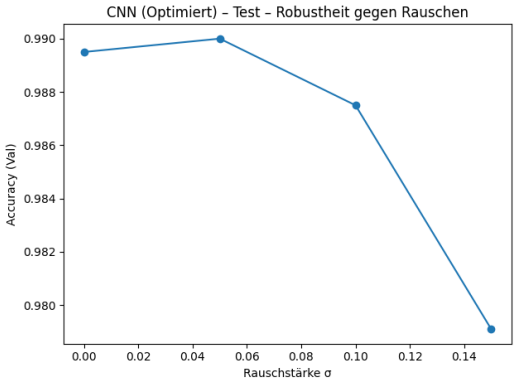

Abbildung 44, Rauschen: CNN Optimiert

Insgesamt bestätigt dieser Test, dass die CNN-Architekturen nicht nur genauer sind, sondern auch deutlich robuster auf gestörte Eingaben reagieren. Besonders das optimierte CNN kombiniert höchste Genauigkeit mit der größten Stabilität gegenüber Störungen, während das MLP bei verrauschten Daten klar an Grenzen stößt.

Nach den einzelnen Vergleichen bietet die Betrachtung der Pareto-Fronts einen abschließenden Gesamtüberblick über die drei Modelle im direkten Vergleich. Hierbei wird die Accuracy jeweils in Relation zu einem Effizienzmaß gesetzt – einmal zur Parameteranzahl, einmal zur Inferenzzeit pro Beispiel und einmal zur Trainingszeit bis zur besten Epoche. So lässt sich erkennen, wie viel Genauigkeit ein Modell im Verhältnis zu seiner Komplexität und seinem Aufwand erreicht.

Die Auswertung zeigt, dass sowohl das Basis- als auch das optimierte CNN in allen drei Diagrammen Teil der Pareto-Front sind. Keines wird vom anderen vollständig dominiert, sodass die Wahl des „besseren“ Modells von den Prioritäten abhängt: Wer maximale Genauigkeit sucht, findet diese im optimierten CNN, während das Basis-CNN weniger rechenintensiv ist und schneller trainiert. Das MLP bildet dagegen in allen Vergleichen das Minimum. Es erreicht die geringste Accuracy, ist aber aufgrund seiner einfachen Architektur besonders effizient in Inferenz und Trainingszeit. In der Analyse Accuracy vs. Parameteranzahl fällt es klar ab, da seine vollständig verbundenen Schichten eine deutlich höhere Anzahl an Parametern erzeugen und es dadurch nicht Teil der Pareto-Front ist (vgl. Abbildungen 45-47).

Zu beachten ist, dass die Aussagekraft der Pareto-Analysen in diesem Fall eingeschränkt ist, da nur drei Modelle verglichen wurden. Schon dadurch, dass ein Modell nicht von beiden anderen gleichzeitig dominiert wird, erscheint es auf der Front. Dennoch verdeutlichen die Ergebnisse, dass es letztlich um eine Abwägung geht: zwischen maximaler Effektivität und effizientem Ressourceneinsatz.

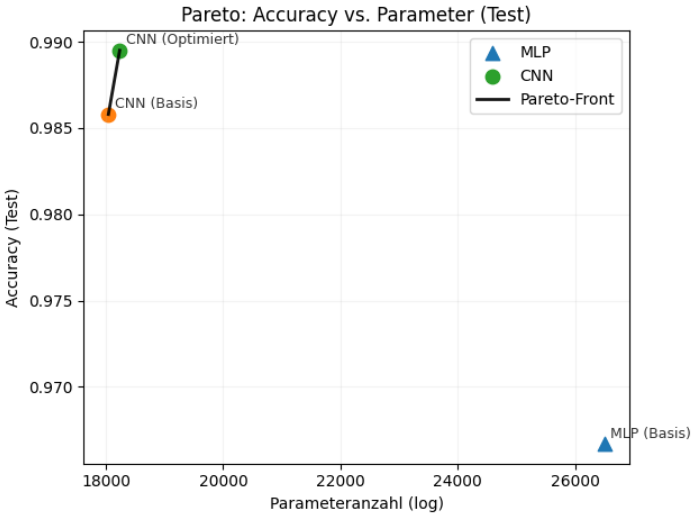

Abbildung 45, Pareto Front: Accuracy vs. Paramter

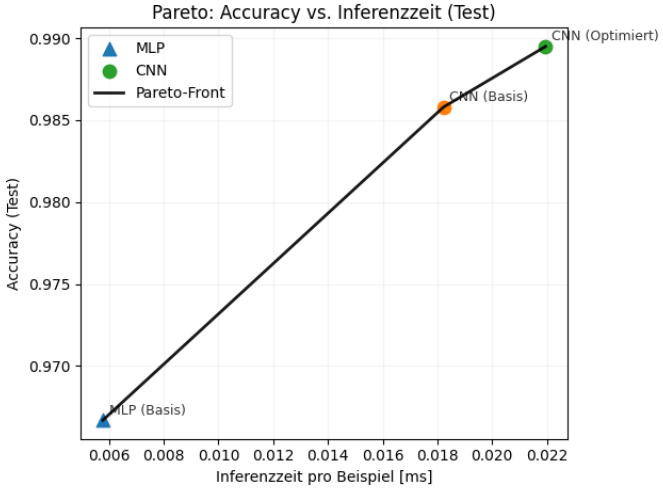

Abbildung 46, Pareto Front: Accuracy vs. Inferenzzeit

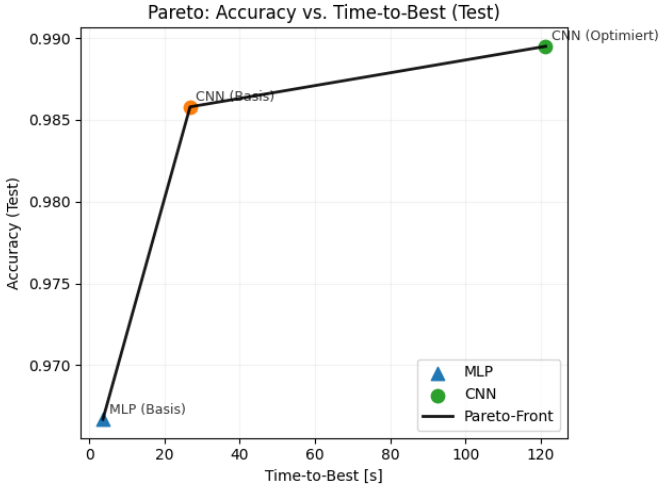

Abbildung 47, Pareto Front: Accuracy vs. Zeit bis zur besten Epoche

Fazit und kritische Reflexion

Es lässt sich abschließend zusammenfassen, dass alle entwickelten Modelle überzeugende Vorhersagen für den MNIST-Datensatz liefern. Besonders bemerkenswert ist, dass bereits das vergleichsweise einfache MLP ohne größere Optimierungen eine solide Genauigkeit von rund 96,7 % erreicht. Deutlich wird jedoch, dass die speziell für die Bildverarbeitung konzipierten CNN-Modelle überlegen sind – neben ihrer sehr hohen Accuracy fallen sie sowohl durch ihre geringere Parameteranzahl als auch durch ihre Robustheit auf. Während die reine Genauigkeit des optimierten CNN im Vergleich zum Basis-CNN nur leicht höher liegt, zeigt sich der eigentliche Vorteil in der Generalisierung und der Widerstandsfähigkeit gegenüber Störungen: Beim Umgang mit verrauschten Eingaben bleibt das optimierte CNN nahezu unbeeindruckt und erzielt selbst bei starker Verzerrung stabile Ergebnisse, während das Basis-CNN hier spürbar nachlässt. Hinzu kommt, dass das optimierte Modell durch die eingesetzten Regularisierungstechniken Overfitting nahezu vollständig vermeidet und eine fast perfekte Generalisierung erreicht. Gleichzeitig bleibt kritisch zu hinterfragen, in welchem Maße sich die zusätzlichen Optimierungen lohnen. Zwar verbessern Dropout und Batch Normalization Stabilität und Robustheit erheblich, sie führen jedoch auch zu einer deutlich längeren Trainingszeit bis zur besten Epoche. Letztlich hängt die Bewertung vom jeweiligen Schwerpunkt ab: maximale Effektivität oder möglichst effiziente Ressourcennutzung. Beide CNN-Modelle liefern insgesamt sehr starke Ergebnisse und bestätigen die Eignung dieser Architektur für Bildklassifikationsaufgaben, das optimierte Modell sticht dabei jedoch in puncto Generalisierung und Robustheit klar hervor.

Dabei gilt es zu berücksichtigen, dass viele Entscheidungen im Optimierungsprozess – etwa zur Architektur oder zu Hyperparametern – knapp und teilweise nach persönlichen Präferenzen getroffen wurden. Andere Wahlmöglichkeiten hätten unter Umständen andere Regularisierungskombinationen oder Batchgrößen begünstigt. Die Ergebnisse sind daher nicht als allgemein gültig zu verstehen, sondern als ein möglicher, konsequent verfolgter Optimierungspfad.

Anhang

Anhang 1: Neuronale Netze – Grundidee

Neuronale Netze bestehen aus vielen miteinander verbundenen Knotenpunkten – sogenannten künstlichen Neuronen –, die in Schichten organisiert sind. Grundlegend haben neuronale Netze folgenden Aufbau:

- Eingabeschicht: nimmt die Rohdaten auf
- Versteckte Schichten: transformieren die Daten, erkennen Muster
- Ausgabeschicht: liefert das finale Ergebnis (z. B. die ermittelte Ziffer)

Jedes Neuron berechnet eine gewichtete Summe seiner Eingaben und wendet darauf eine Aktivierungsfunktion an. Das Training erfolgt über das Backpropagation-Verfahren, bei dem der Fehler rückwärts durch das Netz propagiert wird, um die Gewichte schrittweise zu verbessern. Dieser Vorgang wiederholt sich über mehrere Epochen hinweg, bis das Netz die zugrunde liegenden Muster in den Trainingsdaten ausreichend gelernt hat.

Anhang 2: Multilayer Perceptrons

Ein MLP ist ein neuronales Netz, in dem jede Schicht vollständig mit der nächsten verbunden ist – als versteckte Schichten werden Dense-Schichten eingesetzt. Die Eingabedaten müssen dabei einem eindimensionalen (als flacher Vektor) Format entsprechen. Bilder – wie die des MINST-Datensatz – liegen ursprünglich im 2-D Format vor, weshalb sie entsprechend formatiert werden müssen

Aus einem 2D-Bild mit der Form (28, 28) wird ein 1D-Vektor der Länge 784 (28 * 28 = 784). Dabei ist -1 ein variabler Platzhalter, der dafür sorgt, dass die erste Dimension (Anzahl der Bilder) automatisch passend gewählt wird, alternativ könnte man auch 60 000 für die Trainingsdaten und 10 000 für die Testdaten als fixe Werte schreiben.

```
# Daten in ein 1D-Array umwandeln, da MLPs flache Eingaben erwarten
x_train_flat = x_train.reshape(-1, 28 * 28)
x_test_flat = x_test.reshape(-1, 28 * 28)
```

Ein typisches MLP könnte folgendermaßen aussehen

```
model = Sequential([
    Input(shape=(28 * 28,)),              # Eingabeschicht erwartet 784 Werte (28x28 Pixel)
    Dense(32, activation='relu'),         # erste versteckte Schicht
    Dense(16, activation='relu'),         # zweite versteckte Schicht
    Dense(2, activation='softmax')        # Ausgabeschicht für 2 Klassen
])
```

Die Eingabeschicht erwartet 784 Werte pro Beispiel. Darauf folgen zwei versteckte Dense-Schichten mit 32 bzw. 16 Neuronen, die jeweils eine Aktivierungsfunktion (in diesem Beispiel Rectified Linear Unit (ReLU)) verwenden. (Die verschiedenen Aktivierungsfunktionen und ihre Verwendungen werden in Kapitel 3.4 erläutert). In einer Dense-Schicht empfängt jedes Neuron die Werte aller Neuronen der vorherigen Schicht. Diese Eingabewerte werden jeweils mit einem Gewicht multipliziert, aufaddiert und durch eine Aktivierungsfunktion transformiert – das ergibt den neuen Aktivierungswert des Neurons. Dieser wird wiederum an alle Neuronen der nächsten Schicht weitergegeben, die auf dieselbe Weise ihre Eingaben verarbeiten. Es ist üblich die Anzahl der Neuronen von Schicht zu Schicht zu verringern, um die Merkmalsinformationen schrittweise auf eine entscheidungsfähige Repräsentation zu verdichten. Die Ausgabeschicht besteht beispielsweise aus 2 Neuronen, wobei jedes Neuron eine Klasse repräsentiert (z. B. „wahr“ oder „falsch“ bei binären Klassifikationsproblemen). Hier wird die Softmax-Aktivierungsfunktion eingesetzt.

Häufig wird die Eingabeschicht nicht separat definiert, sondern direkt in der ersten Dense-Schicht über das Argument *input_shape* integriert.

```
model = Sequential([
    Dense(32, activation='relu', input_shape=(28 * 28,)), # Eingabe- und erste versteckte Schicht
    Dense(16, activation='relu'),                         # zweite versteckte Schicht
    Dense(2, activation='softmax')                        # Ausgabeschicht für 2 Klassen
])
```

Ein Nachteil dieser vollständig verbundenen Struktur ist, dass die räumliche Anordnung der Bilddaten verloren geht. Jedes Neuron verarbeitet alle Pixel unabhängig von deren Position, was zu vielen Parametern und hohem Rechenaufwand führt. Lokale Muster – wie benachbarte Pixel – können so nur schwer erkannt werden.

Anhang 3: Convolutional Neural Network

CNNs sind speziell für die Verarbeitung von Bilddaten entwickelt und nutzen deren räumliche Struktur gezielt aus. Im Gegensatz zu MLPs, bei denen jedes Neuron alle Eingabewerte empfängt, arbeiten CNNs mit lokalen Filtern, die auf kleine Bildausschnitte reagieren. Dadurch können sie gezielt lokale Muster wie Kanten, Ecken oder Texturen erkennen – unabhängig davon, wo diese im Bild auftreten.

Damit ein CNN ein Bild korrekt verarbeiten kann, müssen die Eingabedaten im Format eines 3D-Tensors vorliegen: Höhe × Breite × Kanäle (z. B. (28, 28, 1)). Bei Graustufenbildern wie im MNIST-Datensatz ist die Anzahl der Kanäle 1 (für schwarz-weiß), bei RGB-Bildern wären es 3.

```
# CNN erwartet Eingaben in der Form (Anzahl der Bilder, Höhe, Breite, Kanäle)
x_train_cnn = x_train.reshape(-1, 28, 28, 1)  # 1 Kanal für Schwarz-Weiß-Bilder
x_test_cnn = x_test.reshape(-1, 28, 28, 1)
```

Ein typisches CNN könnte folgendermaßen aussehen:

```
model = Sequential([
    Input(shape=(28, 28, 1)),                # Eingabe-Schicht erwartet Bilder mit 28x28 Pixeln und 1 Kanal
    Conv2D(32, (3, 3), activation='relu'),   # erste Convolutional-Schicht
    MaxPooling2D((2, 2)),                    # Max-Pooling-Schicht
    Conv2D(64, (3, 3), activation='relu'),   # zweite Convolutional-Schicht
    MaxPooling2D((2, 2)),                    # Max-Pooling-Schicht
    Flatten(),                               # Flatten-Schicht zum Umwandeln in ein flaches Array
    Dense(64, activation='relu'),            # voll verbundene Schicht
    Dense(10, activation='softmax')          # Ausgabe-Schicht für 10 Klassen
])
```

Der Kern eines CNN besteht aus wiederholten Kombinationen von Faltungsschichten (Convolutional Layers) und Pooling-Schichten, die gemeinsam einen sogenannten Feature-Extraktionsblock bilden.

In einer Faltungsschicht wird das Eingabebild mithilfe kleiner Filter (z. B. 3×3 oder 5×5) verarbeitet. Diese Filter – auch „Kerne“ genannt – gleiten systematisch über das Bild, ähnlich wie ein Fenster, das sich von links nach rechts und oben nach unten über die Pixelbereiche bewegt. An jeder Position berechnet der Filter ein gewichtetes Mittel der Pixelwerte im jeweiligen Ausschnitt. Die entstehenden Ausgabewerte werden zu einer neuen Feature Map zusammengefügt. Eine solche Feature Map gibt an, wo im Bild ein bestimmtes visuelles Merkmal (z. B. eine Kante oder eine Kurve) erkannt wurde und wie stark es dort ausgeprägt ist. Jeder Filter ist dabei auf die Erkennung eines bestimmten Musters trainiert. Wenn beispielsweise 32 Filter verwendet werden, entstehen 32 unterschiedliche Feature Maps – jede für ein eigenes erlerntes Merkmal. In späteren Faltungsschichten wird die Anzahl der Filter häufig erhöht (z. B. von 32 auf 64 oder 128), da dort

komplexere und vielfältigere Muster erkannt werden sollen, die eine feinere Unterscheidung erfordern.

Die Größe der Filter beeinflusst, welche Art von Merkmalen erkannt werden:

- Kleine Filter (z. B. 3×3) fokussieren sich auf feine Details wie einzelne Kanten.
- Größere Filter (z. B. 5×5 oder 7×7) erfassen größere Muster, aber mit weniger Präzision.

Damit Filter auch die Randbereiche des Bildes verarbeiten können, wird oft sogenanntes Padding verwendet. Ohne Padding („valid“) kann der Filter nicht über die äußersten Pixel gleiten, wodurch das Ausgabeformat kleiner wird. Mit Padding („same“) wird das Eingabebild an den Rändern künstlich erweitert, etwa durch Nullen, sodass die Ausgabe die gleiche Höhe und Breite wie die Eingabe hat. Padding sorgt also dafür, dass die Filter alle Bildbereiche gleich behandeln und keine Informationen verloren gehen.

Nach der Faltungsschicht folgt üblicherweise eine Pooling-Schicht, die die erkannten Merkmale verdichtet und irrelevante Details unterdrückt. Dabei wird die Größe der Feature Maps gezielt reduziert, sodass das Netzwerk mit weniger Daten weiterarbeiten kann. Dieser Schritt macht das Modell zudem robuster gegenüber kleinen Verschiebungen im Bild – beispielsweise, wenn eine Linie leicht versetzt dargestellt wird. Die gebräuchlichste Methode ist das Max-Pooling mit einem 2×2-Fenster: Dabei wird aus jedem 2×2-Bereich nur der höchste Wert übernommen – also der Punkt, an dem das erkannte Merkmal am stärksten ausgeprägt war.

Neben Max-Pooling gibt es auch weitere Varianten:

- Average-Pooling berechnet den Durchschnitt aller Werte im Fenster. Es erzeugt glattere Ergebnisse, berücksichtigt aber auch schwache Aktivierungen.
- Global Max-Pooling oder Global Average-Pooling reduzieren jede einzelne Feature Map auf genau einen Wert, indem sie den Maximal- oder Durchschnittswert über die gesamte Map berechnen. Dies wird oft in sehr tiefen Netzen vor der Klassifikation eingesetzt.

Die Fenstergröße beim Pooling muss nicht immer 2×2 sein – größere Fenster führen zu stärkerer Kompression, bergen aber die Gefahr, dass Details verloren gehen.

Die Kombination aus mehreren Faltungs- und Pooling-Schichten ermöglicht es einem CNN, hierarchisch zunehmend komplexe Merkmale zu erfassen:

- In den ersten Schichten erkennt es einfache Muster wie Kanten.

- In späteren Schichten werden daraus Formen und schließlich ganze Objekte.

Nach den Feature-Extraktionsblöcken liegt der Datenoutput als mehrdimensionaler Tensor vor, bestehend aus mehreren Feature Maps. Um die Informationen an die nachfolgenden Schichten weiterzugeben, wird der Tensor durch eine Flatten-Schicht in einen eindimensionalen Vektor umgewandelt. Dabei bleiben alle Werte erhalten, jedoch geht die räumliche Anordnung der Merkmale verloren. Das ist in diesem Netzwerkabschnitt unproblematisch, da Dense-Schichten alle Eingabewerte gleichberechtigt und unabhängig von ihrer Position verarbeiten.

Es folgen nun eine oder mehrere vollverbundene Dense-Schichten, wie man sie aus MLPs kennt. Sie kombinieren die extrahierten Merkmale und verdichten diese oft schrittweise in abnehmender Neuronenzahl, um aus vielen Detailinformationen wenige aussagekräftige Merkmalskombinationen zu formen. So wird eine globale Entscheidung getroffen – beispielsweise darüber, welche Ziffer dargestellt wird. Die letzte Schicht ist wie bei einem MLP eine Dense-Ausgabe mit Softmax-Aktivierungsfunktion.