SEGMENTIERUNG NEURONALER NETZWERKE:

Eine Strategie zur Beschleunigung des Trainingsprozesses?

Alessio Maffioletti (Jg. 06)

Maturitätsarbeit and der Kantonsschule Uster

Betreuerin: Seung Hee Ma

Expertin: Theresa Luternauer

08. Januar 2025

Inhaltsverzeichnis

Abbildungsverzeichnis

1. Einleitung

1.1 Motivation

Mit der stetig wachsenden Nachfrage nach KI-Modellen steigt auch der Bedarf an Rechenleistung. Selbst für kleinere Modelle, wird oft viel Rechenkapazität benötigt, um die Trainingszeiten in einem akzeptablen Rahmen zu halten. Bei größeren Anwendungen ist die Rechenleistung ein oft unvermeidlicher Aufwand, aber für kleinere, individuelle Projekte wird sie schnell zum begrenzenden Faktor.

Mein Ziel ist es, eine Methode zu entwickeln, die diesen Aufwand reduziert, sodass kleinere Modelle einfacher und kostengünstiger trainiert werden können. Dadurch könnten persönliche Projekte und kleinere Anwendungen effizienter umgesetzt werden, ohne dass hohe Rechenleistung erforderlich ist. Diese Optimierung würde die Zugänglichkeit von KI-Technologien erweitern und deren Nutzung auch im kleineren Rahmen fördern.

1.2 Fragestellung

In dieser Arbeit untersuche ich dir folgende Frage: Wie wirkt sich die Segmentierung eines neuronalen Netzwerks auf die Effizient und Trainingszeit bei Bildklassifizierungsaufgaben aus, und wie schneidet dieser Ansatz im Vergleich zu einem nicht segmentierten Modell ab?

1.3 Vorgehen

Zur Beantwortung dieser Frage erstelle ich einen Datensatz, der auf einem bestehenden Bild-Datensatz basiert, jedoch die Komplexität erhöht: Es wird pro Bild eine Zahl auf einem schwarzen Hintergrund mit zufälliger Position platziert. Ich trainiere daraufhin zwei unterschiedliche Modelle mit Python, diese Zahl zu erkennen. Das erste Modell, das SingleNet, klassifiziert das Motiv direkt anhand der Bilddaten. Das zweite Modell, das DualNet, hingegen ist in zwei neuronale Netzwerke aufgeteilt: Das erste Netzwerk, das Koordinatennetz, bestimmt die Position des Motivs im Bild, und das zweite, das Klassifikationsnetz, verwendet diese Position als zusätzlichen Input zur Bildklassifizierung.

Um die Effizienz beider Ansätze zu vergleichen, wird untersucht, wie lange beide Modelle benötigen, um eine Trefferquote von über 90 % zu erreichen.

2.Theorie

2.1 Neuronale Netzwerke

Neuronale Netzwerke sind Modelle des maschinellen Lernens, die aus Layers (unten) von Neuronen bestehen, die miteinander verbunden sind und numerische Gewichtungen besitzen. Während des Trainings werden diese Gewichte angepasst, um den Fehler zwischen den Vorhersagen und den tatsächlichen Ergebnissen zu minimieren. In meiner Arbeit liegt der Fokus auf der Optimierung von Netzwerkstrukturen, insbesondere im Hinblick auf die Effizienz und Trainingszeit von Modellen, die auf Bildklassifizierungsaufgaben angewendet werden.

2.2 Rechenleistung

Die Gewichte des Neuronalen Netzwerks bestimmen direkt den Output, also je mehr Gewichte das Netzwerk hat, desto komplexere Zusammenhänge zwischen den Eingabedaten kann das Netzwerk finden. Jedoch muss jedes Gewicht optimiert werden. Das heisst, dass die Rechenoperation beim Training des Netzwerks mehr Leistung und somit mehr Zeit braucht. Das Ziel ist es die richtige Balance zu finden zwischen Komplexität des Netzwerks und Trainingszeit. Wenn man ein einfaches Datenset hat, macht es keinen Sinn ein komplexes Netzwerk darauf zu trainieren, da es zu lange dauern würde einen sinnvollen Output zu bekommen. Im Gegensatz, wenn das Datenset komplex ist und das Netzwerk zu klein, wird das Netzwerk keine guten Werte rausgeben, da es nicht genug Gewichte zum Optimieren hat. Man kann das mit dem fitten einer Kurve mit einer Binomialfunktion vergleichen; je höher das Grad der Funktion, desto genauer wird die Kurve approximiert.

2.3 Dense Layers

In der Welt des maschinellen Lernens und der neuronalen Netzwerke gibt es viele verschiedene Layer-Typen, die in einem Netzwerk Werte auf unterschiedliche Weise berechnen. Der einfachste Layer, der Dense Layer, besteht aus n Neuronen. Dieser Layer besitzt zwei Parameter, die während des Trainings angepasst werden: einen Vektor mit n Gewichten und ein Bias. Zusätzlich verwenden alle Layer eine Aktivierungsfunktion. Diese entscheidet, ob der Neuron aktiviert wird und führt so Nichtlinearität ein. Der Output eines Dense Layers wird berechnet, indem der Input-Vektor wie folgt verarbeitet wird:

$$Output = Aktivierungsfunktion(Skalarprodukt(Input, Gewichtsvektor) + Bias)$$

(Keras API, 2024)

2.4 Convolutional Layers

Wenn ein neuronales Netzwerk mit Bildern trainiert wird, werden häufig Convolutional Layers verwendet. Im Unterschied zu Dense Layers nutzen sie statt eines Vektors eine Matrix. Dadurch können Inputs mit höheren Dimensionen, wie beispielsweise Bilder (zweidimensionale Daten), effizient verarbeitet werden. Die Berechnung erfolgt analog:

$$Output = Aktivierungsfunktion(dot(Input, Gewichtmatrix) + Bias)$$

wobei $dot()$, eine Matrixmultiplikation beschreibt. (Tensorflow APi, 2024).

Wichtig ist, dass bei der Kombination von Convolutional Layers und Dense Layers in einem Netzwerk die Dimensionen der Outputs aus den Convolutional Layers angepasst werden müssen, da Dense Layers mit Vektoren arbeiten. Diese Anpassung erfolgt durch einen Flatten Layer, der die mehrdimensionalen Outputs der Convolutional Layers in einen Vektor umwandelt. Dies sorgt dafür, dass die Dimensionen kompatibel sind und das Training korrekt durchgeführt werden kann (Keras API, 2024) .

2.5 GlobalAveragePooling2d Layer

Der Flatten Layer wandelt alle Parameter der Input-Matrix in einen einzigen Vektor um. Der nächste Layer erhält somit einen großen Vektor als Input. Um die Größe dieses Vektors zu reduzieren, kann stattdessen ein GlobalAveragePooling2D Layer verwendet werden. Dieser berechnet den Durchschnitt der Werte in einer zweidimensionalen Matrix (z. B. einem Bild) und erzeugt daraus einen kompakten Vektor (saturncloud, 2024).

Dadurch werden zwar nicht alle Daten weitergegeben, aber die aussagekräftigsten Informationen bleiben erhalten, und die Weiterverarbeitung erfolgt mit kleineren Vektoren. Dennoch wird dieser Layer nicht in allen Fällen verwendet, da er je nach Anwendung nicht immer sinnvoll ist.

2.6 Dropout Layers

Ein Dropout Layer dient im neuronalen Netzwerk als Regulierer, um Überanpassung (Overfitting) zu vermeiden. Dieser Layer setzt einen festgelegten Prozentsatz der Gewichte in einem Vektor zufällig auf 0 – dieser Prozentsatz wird als Dropout-Rate bezeichnet.

Die übrigen Gewichte, die nicht auf 0 gesetzt wurden, werden mit dem Faktor $\frac{1}{(1-DropoutRate)}$ multipliziert (Keras API, 2024). Dadurch bleibt die Summe der Gewichte konstant und das Netzwerk kann weiterhin stabil trainiert werden.

2.7 Bayes'sche Optimierung

Bayes'sche Optimierung ist eine Methode, um Funktionen zu optimieren, bei denen man nicht weiß, wie die Funktion aussieht. Das sind sogenannte Black-Box-Funktionen. Diese Optimierungsmethode basiert darauf, zunächst zufällige Werte als Parameter für die Funktion auszuwählen und den Lösungswert zu messen. Basierend auf diesen Messungen wird ein probabilistisches Modell erstellt, meist ein Gauß-Prozess (Wikipedia, 2024), der schätzt, wie die Funktion in anderen Bereichen aussehen könnte. Mit Hilfe dieses Modells wählt die Bayes'sche Optimierung dann gezielt neue Parameter aus, die entweder die beste Lösung verbessern könnten (Exploration) oder die bisherigen besten Werte weiter verfeinern (Exploitation). Durch diesen Wechsel zwischen Ausprobieren und Verfeinern kann die beste Lösung mit möglichst wenigen Messungen gefunden werden.

Da ein neuronales Netzwerk zwar eine bekannte Funktion ist, aber zu komplex für eine direkte Berechnung der besten Parameter, kann es als Black-Box-Funktion betrachtet werden. Somit eignet sich die Bayes'sche Optimierung auch zur Optimierung von neuronalen Netzwerken. Dieser Vorgang ist im maschinellen Lernen sehr wichtig und wird Hyperparameteroptimierung genannt.

2.8 One-Hot-Codierung

One-Hot-Codierung wird verwendet, um Kategorien in Zahlenwerte umzuwandeln, sodass sie im Neuronalem Netzwerk alle eine gleiche Gewichtung haben. Bei der Klassifizierung von Zahlen ist es ungünstig die Zahlen durch ihren Zahlenwert zu repräsentieren, da so höhere Zahlen vom Netzwerk bevorzugt werden könnten. So kommt es zu einer ungenauen und falschen Auswertung. Mit One-Hot-Codierung werden die Kategorien durch eine List repräsentiert, bei der Jede Stelle einer Kategorie entspricht. Beispiel, die Ziffer 3 als One-Hot-Codierung dargestellt: $[0, 0, 0, 1, 0, 0, 0, 0, 0, 0]$.

2.9 Batchgrösse

Beim Training eines neuronalen Netzwerks kann der Datensatz in kleinere Stücke (Batches) unterteilt werden, und das Training erfolgt dann mit diesen einzelnen Batches. Dadurch muss nicht der gesamte Datensatz im Arbeitsspeicher gespeichert werden, was es ermöglicht, auch größere Datensätze mit begrenztem Arbeitsspeicher zu verwenden. Nach jedem Ablauf eines Batches werden die Gewichte im Netzwerk angepasst. Das bedeutet, dass bei kleineren Batches die Gewichte häufiger angepasst werden, was zu einer höheren Trainingszeit führen kann. (geeksforgeeks, 2025)

3. Methodik

3.1 Datensatzerstellung

Da die Modelle eine höhere Datensatzkomplexität benötigen, wurde der MNIST-Datensatz entsprechend angepasst. Der MNIST-Datensatz besteht aus 60'000 Bildern mit handgeschriebenen Zahlen. Die Bilder sind haben eine Auflösung von 32*32 Pixel und 3 Farbkanälen (RGB). Da aber die Farben keine Entscheidende Rolle für die Klassifizierung der Zahl spielen, werde die Bilder einfarbig genutzt (Grauton). Um die Komplexität zu erhöhen, werden die Bilder mit zufälligen Koordinaten auf einen schwarzen Hintergrund, der eine Auflösung von 128*128 Pixel hat, platziert (Abbildung 1).

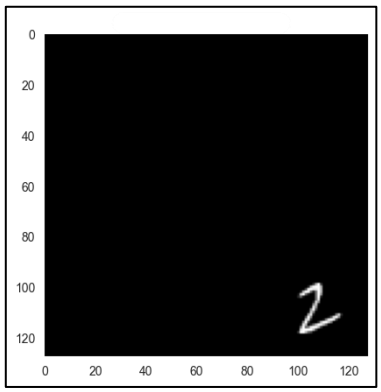


Abbildung 1: Beispiel aus dem angepasstem Datensatz

Die Koordinaten der Zahl werden zwar zufällig gewählt, sind aber im Code gespeichert, also kann der Datensatz so angepasst werden, sodass nicht nur die einzelnen Pixel des Bildes enthalten sind, sondern auch die Koordinaten. Mit diesem Datensatz ist es möglich beide Modelle zu trainieren. Das SingleNet, bekommt beide Inputs aus dem Datensatz: die Pixelwerte im Bild und die Koordinaten. Als Output wird es die Zahl, welche auf dem Bild ist, One-Hot-Codiert (oben) ausgeben. Diese Zahl ist auch im Datensatz erhalten, als Label, und wird so korrekt im Training eingesetzt. Das DualNet, besteht, wie schon erwähnt, aus zwei Neuronalen Netzwerken, dem Koordinatennetz und dem Klassifikationsnetz. Das Koordinatennetz kann jetzt mithilfe dieses Datensatzes schon trainiert werden. Jedoch im Gegensatz zum SingleNet, wird es die Koordinaten der Zahl vorhersagen (Abbildung 2).

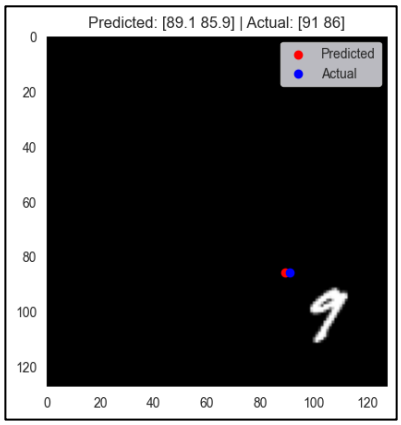
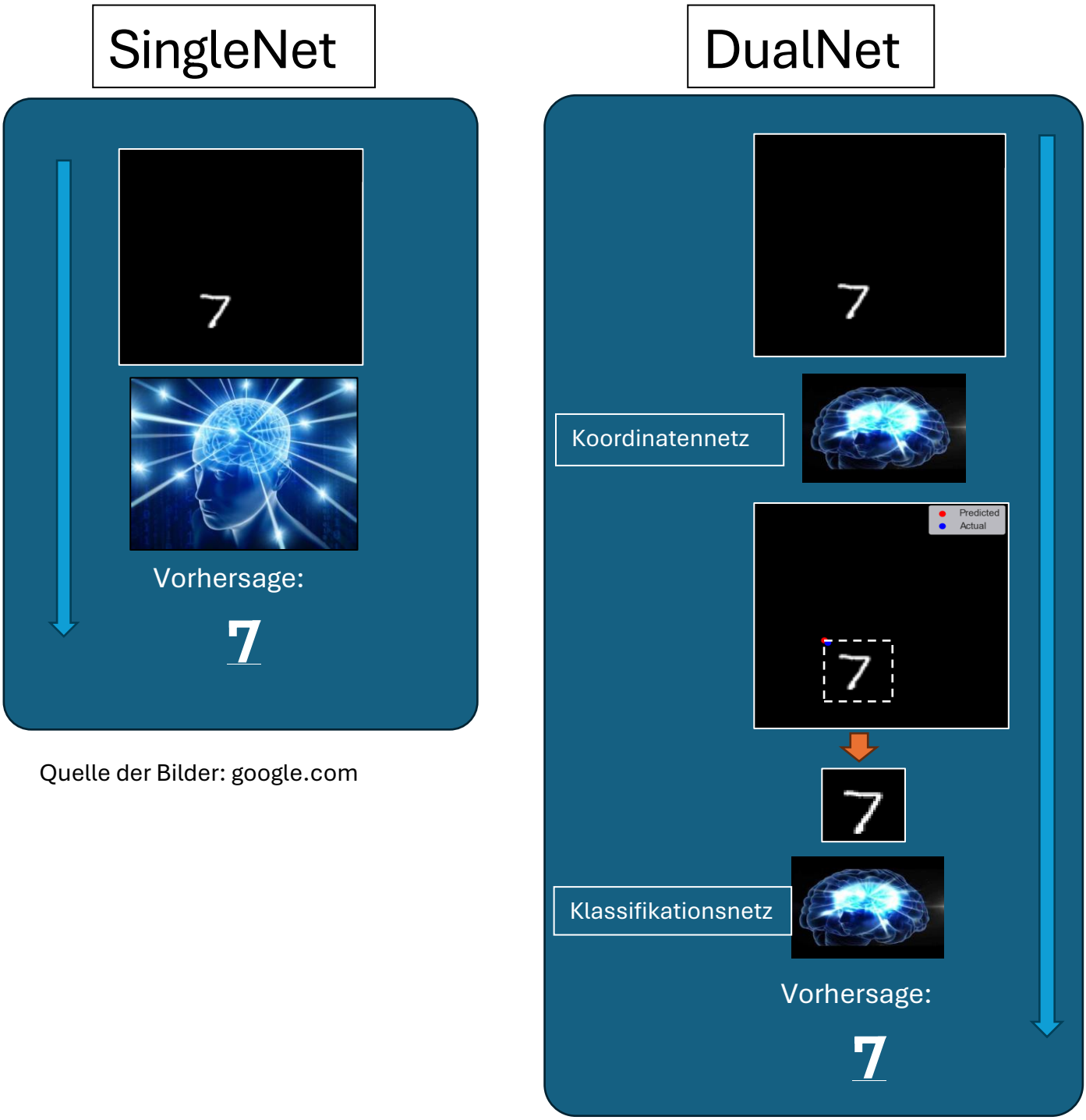


Abbildung 2: Vorhergesagte Koordinaten vom Koordinatennetz

Dementsprechend muss der Datensatz leicht angepasst werden, sodass nur die Pixelwerte als Input gegeben werden, und die Koordinaten als Label. Die Zahl selbst wird noch nicht vorhergesagt. Im nächsten Schritt wird ein neuer Datensatz erstellt. Es werden die Koordinaten mit dem Koordinatennetz für alle Bilder vorhergesagt und das Bild wird zugeschnitten, sodass die Zahl im Bild zentriert ist. So wird das Bild kleiner, hat also weniger Pixelwerte, und hat nicht zufällige Koordinaten. Der neue Datensatz besteht jetzt aus 60'000 zugeschnittenen Bildern als Input und der Zahl selbst als Label. Zusätzlich

bietet der MNIST-Datensatz einen Evaluierenden Datensatz. Dieser wird nicht im Training verwendet, sondern in der Auswertung. Dies verhindert, dass das Netzwerk die Werte einfach auswendig lernt (Overfitting). Der Evaluierungsdatensatz besteht aus 10’000 Bildern und wird für die korrekte Auswertung analog zum Trainingsdatensatz angepasst.

3.2 Visuelle Repräsentation des Vorganges



Quelle der Bilder: google.com

Abbildung 3: Visuelle Repräsentation des Vorganges

3.3 Netzwerkstrukturen

Da jedes der Neuronalen Netzwerke in den zwei Modellen, Bilder als Input nehmen, werden bei allen Netzwerken eine ähnliche Struktur verwendet. Sie bestehen aus einer Anzahl Convolutional Layers, einer Flatten Funktion und anschliessend einer Anzahl Dense Layers. Die genaue Anzahl der Layer wird noch nicht fixiert, weil diese noch beim Optimierungsschritt bestimmt werden. Zusätzlich werden zwischen den Dense Layers, Dropout Layers verwendet, um Überanpassung zu minimieren. Die Netzwerke unterscheiden sich jedoch in den Input- und Output-Grössen: Da das SingleNet ein Bild mit 128*128 Pixeln und den Koordinaten der Zahl als Input nimmt, hat es zwei Input Layers: einer mit Grösse (128,128) und einer mit Grösse (2). Das SingleNet klassifiziert die One-Hot-Codierte(oben) Ziffer und hat somit einen Output von (10).

Das Koordinatennetz nimmt hingegen nur das Bild als Input und sagt Koordinaten vorher. Also hat es eine Input-Grösse von (128,128) und eine Output-Grösse von (2).

Das Klassifikationsnetz nimmt das Bild, das mit einem Puffer entsprechend der Abweichung des Koordinatennetzes zugeschnitten wurde, als Eingabe und klassifiziert die Ziffer. Also hat es eine Input-Grösse von (32+Puffer, 32+Puffer) und eine Output-Grösse von (10).

3.4 Training der Modelle

Um zu vergleichen, welches Modell effizienter ist, werden alle Netzwerke separat trainiert und so weit optimiert, bis ein bestimmter Zielwert erreicht wird. Das SingleNet und das Klassifizierungsnetz werden trainiert, bis eine Trefferquote von mindestens 90 % erreicht ist. Das Koordinatennetz zielt auf eine Abweichung von 1.5 Pixel, jedoch wird diese Abweichung am Schluss von jeder Epoche gemessen, sodass die endgültige Abweichung darunter liegen kann. Somit ist nicht die Ideale Abweichung wichtig für die nächsten Schritte im Training, sondern die endgültige Abweichung, welche das Netzwerk erreicht hat. Anschließend werden die besten Trainingszeiten gespeichert und ausgewertet.

3.5 Optimierung eines Modelles

Um die besten Hyperparameter eines Modells zu finden und somit auch die kürzeste Trainingszeit zu ermitteln, muss eine Optimierung durchgeführt werden (oben).

Um die Hyperparameteroptimierung richtig durchzuführen, wurde entschieden, eine bereits bestehende und bewährte Optimierungsfunktion aus dem Bereich des maschinellen Lernens zu verwenden. Für mein Problem eignete sich die Bayes'sche Optimierung am besten (oben).

Zur Implementierung wurde ein Python-Modul namens Optuna genutzt, was den Prozess erheblich vereinfachte. Um die besten Hyperparameter für ein Netzwerk zu finden, wurden nur die Grenzen für die minimale und maximale Größe sowie die Anzahl der Layer definiert. Anschließend wurde das Netzwerk mit den von der Optimierungsfunktion ausgewählten Parametern trainiert.

Die Trainingszeit wurde dabei als Penalty (Strafwert) an die Optimierungsfunktion übergeben. Diese berechnet anhand des Penalty-Werts, welche Parameter mit hoher Wahrscheinlichkeit die besten sind. Der Prozess wird beliebig oft wiederholt, bis sich die Trainingszeit nicht mehr signifikant verbessert.

Um zu verhindern, dass ein Netzwerk zu lange trainiert wird, wird die Trefferquote nach Ablauf der kürzesten Trainingszeit gemessen. Falls die gewünschte Trefferquote nicht erreicht wurde, wird die Penalty durch die erreichte Trefferquote geteilt. Auf diese Weise wird ein Netzwerk mit niedriger Trefferquote stärker bestraft. Beim Koordinatennetz wird die Penalty zusätzlich mit der Abweichung multipliziert, sodass der Prozess analog funktioniert.

4. Ergebnisse

4.1 Numerische Trainingsergebnisse

Es wurden insgesamt zwei Durchläufe durchgeführt. Diese hatten identische Anfangsbedingungen, unterschieden sich jedoch darin, dass im zweiten Durchlauf das Koordinatennetz eine Abweichung von 5 statt 1.5 Pixel anstrebte.

4.1.1 Durchlauf 1

Nach der Optimierung wurden die Ergebnisse der einzelnen Netzwerke als JSON-Dateien gespeichert. Die Parameter der besten Trainingsdurchläufe aus dem ersten Durchlauf lauten:

SingleNet	Koordinatennetz	Klassifikationsnetz

<pre>"training_time": 17.3,
"trial_number": 143,
"params": {
"num_conv_layers": 5,
"num_dense_layers": 1,
"conv_0_size": 4,
"conv_1_size": 18,
"conv_2_size": 19,
"conv_3_size": 39,
"conv_4_size": 49,
"dense_0_size": 71,
"flatten_type":
"global_average",
"activation": "relu",
"optimizer": "adam"}</pre>
	<pre>"training_time": 17.9,
	"trial_number": 239,
	"params": {
	"num_conv_layers": 5,
	"num_dense_layers": 1,
	"conv_0_size": 4,
	"conv_1_size": 8,
	"conv_2_size": 24,
	"conv_3_size": 175,
	"conv_4_size": 147,
	"dense_0_size": 210,
	"flatten_type":
	"global_average",
	"activation": "relu",
	"optimizer": "adam"}</pre>
		<pre>"training_time": 3.6,
		"trial_number": 177,
		"params": {
		"num_conv_layers": 3,
		"num_dense_layers": 1,
		"conv_0_size": 4,
		"conv_1_size": 4,
		"conv_2_size": 91,
		"dense_0_size": 93,
		"flatten_type":
		"global_average",
		"activation": "relu",
		"optimizer": "adam"}</pre>

Aus dieser Datei lässt sich ablesen, dass die beste Trainingszeit für das SingleNet 17.3 Sekunden, für das Koordinatennetz 17.9 Sekunden und für das Klassifikationsnetz 3.6 Sekunden beträgt. Damit ist das SingleNet in diesem Fall um 4.2 Sekunden schneller als das DualNet.

Zusätzlich enthalten die Ergebnisse den Parameter «trial_number», der angibt, nach wie vielen Optimierungsschritten die jeweilige Trainingszeit erreicht wurde. So ist erkennbar, dass das SingleNet nach 143, das Koordinatennetz nach 239 und das Klassifikationsnetz nach 177 Optimierungen die besten Trainingszeiten erzielten.

4.1.2 Durchlauf 2

Folgende sind die Ergebnisse des zweiten Durchlaufs.

SingleNet	Koordinatennetz	Klassifikationsnetz

<pre>"training_time": 17.5,
"trial_number": 310,
"params": {
"num_conv_layers": 5,
"num_dense_layers": 5,
"conv_0_size": 4,
"conv_1_size": 10,
"conv_2_size": 17,
"conv_3_size": 171,
"conv_4_size": 52,
"dense_0_size": 24,
"dense_1_size": 12,
"dense_2_size": 125,
"dense_3_size": 129,
"dense_4_size": 219,
"flatten_type":
"global_average",
"activation": "relu",
"optimizer": "adam"}</pre>
	<pre>"training_time": 13.8,
	"trial_number": 287,
	"params": {
	"num_conv_layers": 5,
	"num_dense_layers": 4,
	"conv_0_size": 4,
	"conv_1_size": 4,
	"conv_2_size": 20,
	"conv_3_size": 36,
	"conv_4_size": 10,
	"dense_0_size": 194,
	"dense_1_size": 217,
	"dense_2_size": 164,
	"dense_3_size": 16,
	"flatten_type":
	"global_average",
	"activation": "relu",
	"optimizer": "adam"}</pre>
		<pre>"training_time": 3.5,
		"trial_number": 400,
		"params": {
		"num_conv_layers": 3,
		"num_dense_layers": 1,
		"conv_0_size": 4,
		"conv_1_size": 9,
		"conv_2_size": 74,
		"dense_0_size": 68,
		"flatten_type":
		"global_average",
		"activation": "relu",
		"optimizer": "adam"}</pre>

Diese Ergebnisse zeigen, dass das SingleNet eine Trainingszeit von 17.5 Sekunden, das Koordinatennetz eine Trainingszeit von 13.8 Sekunden und das Klassifikationsnetz eine Trainingszeit von 3.5 Sekunden erreicht haben. Damit beträgt die Gesamttrainingszeit des DualNet 17.3 Sekunden und ist im Vergleich zum SingleNet um 0.2 Sekunden schneller.

Aus dem Parameter «trial_number» geht außerdem hervor, dass das SingleNet nach 310, das Koordinatennetz nach 287 und das Klassifikationsnetz nach 400 Optimierungen die besten Trainingszeiten erzielten.

Zusätzlich wurden Parameter wie «flatten_type», «activation» und «optimizer» aufgeführt. Diese Parameter sind in diesem Versuch jedoch nicht aussagekräftig, da sie festgelegt wurden und nicht Teil des Optimierungsprozesses waren.

4.2 Netzwerkstrukturen

In der Liste «params» aus den Ergebnissen sind die Parameter festgehalten, mit denen das Netzwerk aufgebaut wurde. Die Anzahl der Convolutional und Dense Layers wird durch die Parameter «num_conv_layers» und «num_dense_layers» angezeigt, wobei die Größe der Layers mit den Parametern
«conv_n_size» und «dense_n_size» angegeben wird.
Um die Größen besser zu visualisieren, wurde eine Grafik erstellt, die die Anteile, der Layers in Prozent anzeigt. Jedes Kuchenstück der Kuchengrafik stellt einen Layer dar, wobei die blauen Kuchenstücke die Convolutional Layers und die roten Stücke die Dense Layers repräsentieren.

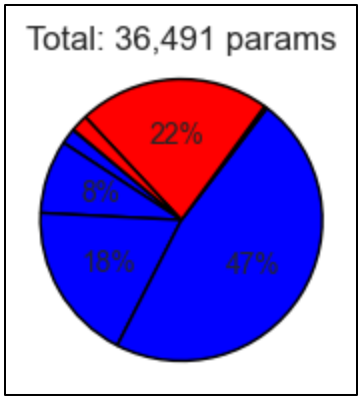


Abbildung 6: Layer-Verteilung vom SingleNet, 1. Durchlauf

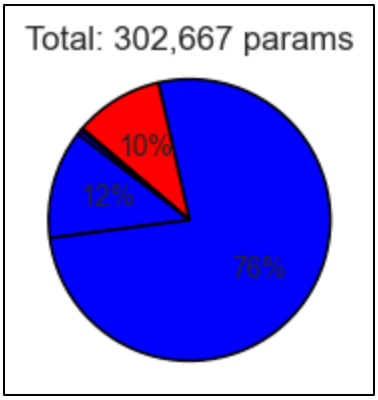


Abbildung 5: Layer-Verteilung vom Koordinatennetz, 1. Durchlauf

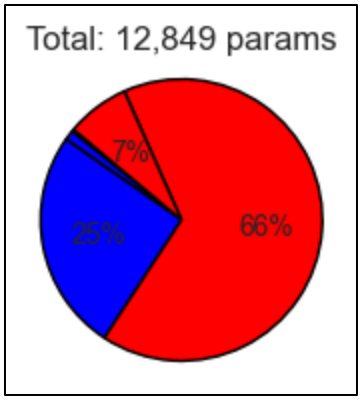


Abbildung 4: Layer-Verteilung vom Klassifikationsnetz, 1. Durchlauf

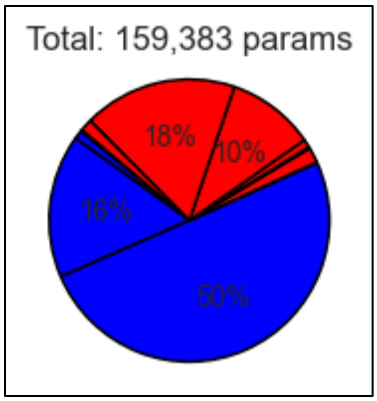


Abbildung 8: Layer-Verteilung vom SingleNet 2. Durchlauf

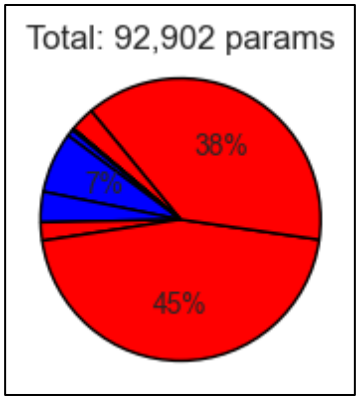


Abbildung 9: Layer-Verteilung vom Koordinatennetz 2. Durchlauf

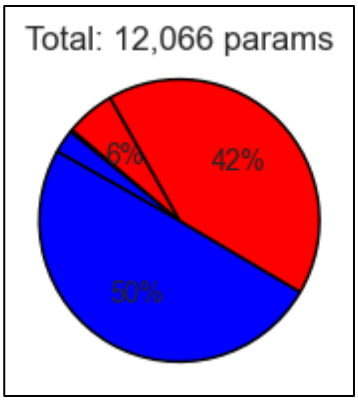


Abbildung 7: Layer-Verteilung vom Klassifikationsnetz 2. Durchlauf

Es ist deutlich zu erkennen, dass sich die Netzwerke zwischen den beiden Durchläufen stark unterscheiden. Im ersten Durchlauf bestand das SingleNet zu etwa drei Vierteln aus Convolutional Layers und hatte eine Gesamtanzahl von 36'491 Parametern (interne Gewichte der Layers). Das Koordinatennetz bestand zu fast 90% aus Convolutional Layers und verfügte über insgesamt 302'667 Parameter. Das Klassifikationsnetz hingegen bestand zu etwa drei Vierteln aus Dense Layers und hatte 12'849 Parameter. Im Vergleich zum DualNet hatte das SingleNet im ersten Durchlauf somit fast 280’000 Parameter weniger. Im zweiten Durchlauf veränderte sich die Zusammensetzung der Netzwerke deutlich: Das SingleNet hatte einen Anteil an Convolutional Layers von etwa 66%, während das Koordinatennetz nun über 83% Dense Layers aufwies. Das Klassifikationsnetz zeigte eine eher ausgeglichene Verteilung zwischen Dense und Convolutional Layers, mit einem Verhältnis von etwa 50/50.

Auffällig ist zudem, dass das SingleNet im zweiten Durchlauf erheblich mehr Parameter hatte. Mit insgesamt 159'383 Parametern verfügte es im Vergleich zum DualNet, das insgesamt 104'968 Parameter hatte, über etwa ein Drittel mehr Parameter.

4.3 Endgültige Metriken

Zusätzlich zu den Ergebnissen der Optimierung können auch die endgültigen Metriken der Netzwerke gemessen werden. Für den ersten Durchlauf beträgt,

- die Endtrefferquote des SingleNet beträgt 0.9139 (ca. 91.4%).
- die Endabweichung des Koordinatennetzes beträgt 1.344091 (ca. 1.3 Pixel).
- die Endtrefferquote des Klassifikationsnetzes beträgt 0.9086 (ca. 90.9%).

Für den zweiten Durchlauf beträgt,

- die Endtrefferquote des SingleNet beträgt 0.9022 (ca. 90.2%).
- die Endabweichung des Koordinatennetzes beträgt 2.7740 (ca. 2.8 Pixel).
- die Endtrefferquote des Klassifikationsnetzes beträgt 0.9012 (ca. 90.1%).

4.4 Trainingsgraphen

1. Graph SingleNet

Während des Optimierungsprozesses wurde der Verlauf des Trainings nur epochenweise gespeichert, da eine batchweise Speicherung zu viel Rechenleistung beansprucht hätte. Da die meisten Netzwerke bereits innerhalb von zwei Epochen vollständig trainiert waren, sind Graphen, die auf den tatsächlichen Trainingsergebnissen basieren, nicht aussagekräftig.

Stattdessen können die Verläufe simuliert werden. Diese simulierten Graphen basieren jedoch nicht auf den echten Trainingsdaten und dienen lediglich zur Veranschaulichung. Der simulierte Verlauf des SingleNet in beiden Durchläufen sieht wie folgt aus:

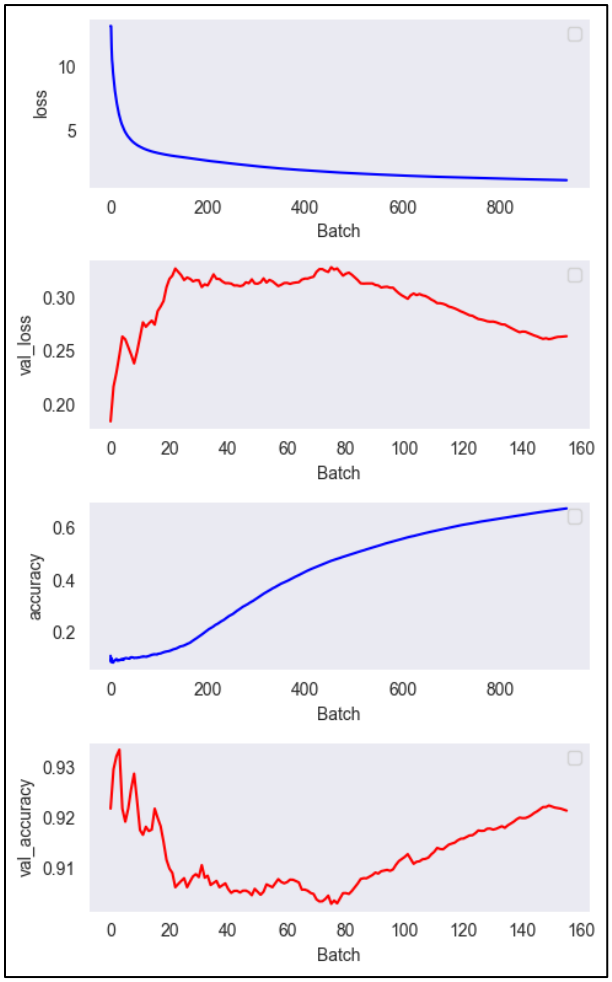


Abbildung 11: simulierter Verlauf verschiedener Metriken im Training vom SingleNet, im 1. Durchlauf

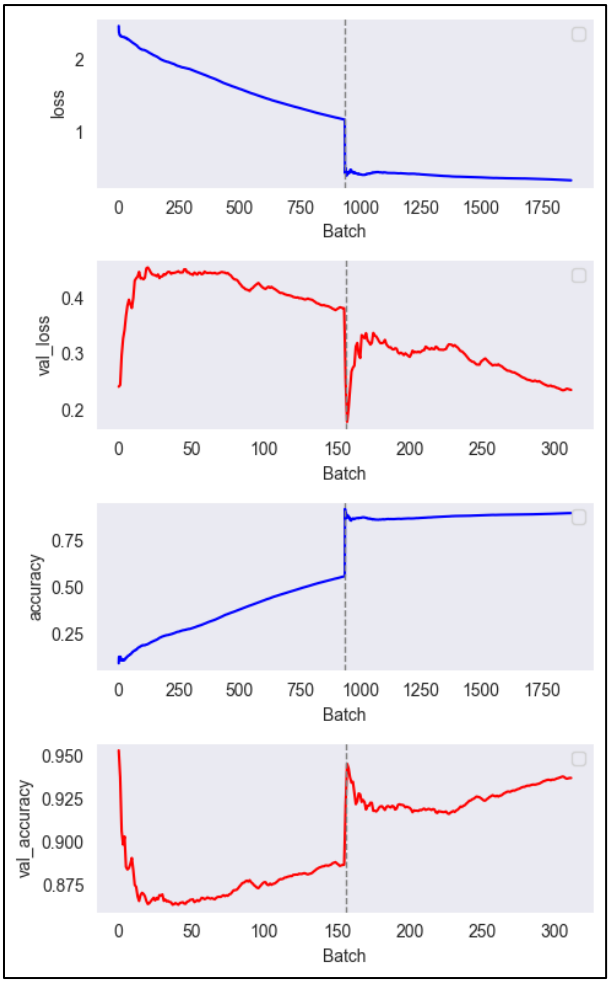


Abbildung 11: simulierter Verlauf verschiedener Metriken im Training vom SingleNet, im 2. Durchlauf

Es sind vier Liniengraphen zu erkennen, jeweils für jeden Durchlauf. Die X-Achse ist mit «Batch» beschriftet, die Y-Achse zeigt die Metriken «loss», «val_loss», «accuracy» und «val_accuracy». Die X-Achse gibt den aktuellen Batch des Datensets an, während die Y-Achse jeweils eine gemessene Metrik darstellt. Der «loss» beschreibt den Verlustwert

während des Trainings, die «accuracy» die Trefferquote. Die Metriken «val_loss» und «val_accuracy» beziehen sich auf den Verlustwert und die Trefferquote, die auf dem Evaluierungsdatenset gemessen wurden.
Die Graphen zeigen die Entwicklung der jeweiligen Metrik nach jedem Batch. Die gestrichelte Linie, die in der Mitte sichtbar ist, markiert das Ende einer Epoche. In der Simulation ist zu sehen, dass das Netzwerk im zweiten Durchlauf über zwei Epochen trainiert wurde.
Auffällig ist, dass zu Beginn der zweiten Epoche alle Metriken einen deutlichen Sprung in Richtung Verbesserung zeigen.

2. Graph Klassifikationsnetz
Der simulierte Verlauf der Trainings, vom Klassifikationsnetz sieht wie folgt aus:

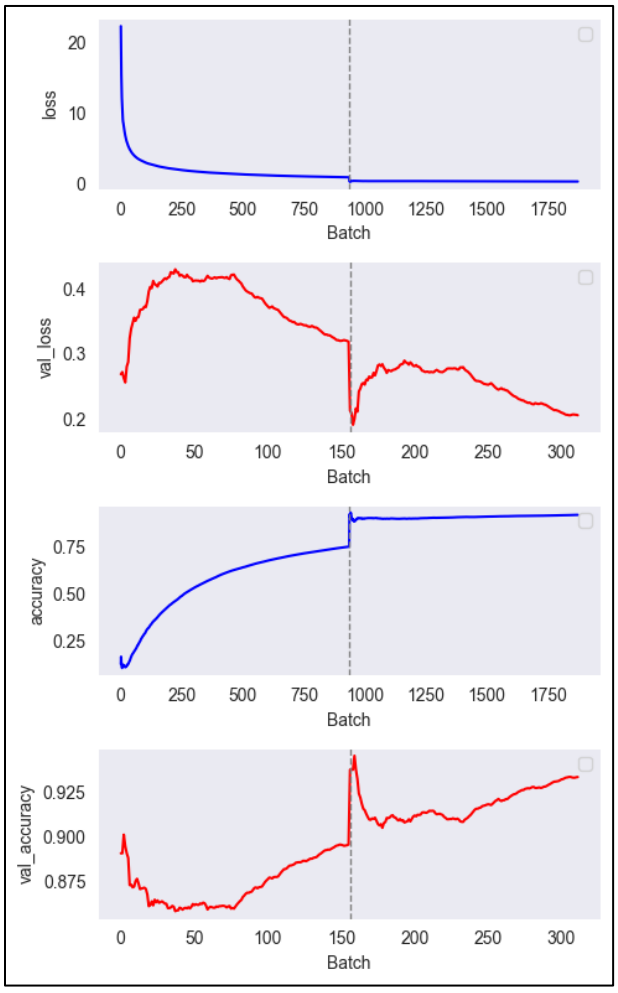


Abbildung 13: simulierter Verlauf verschiedener Metriken im Training vom Klassifikationsnetz, im 1. Durchlauf

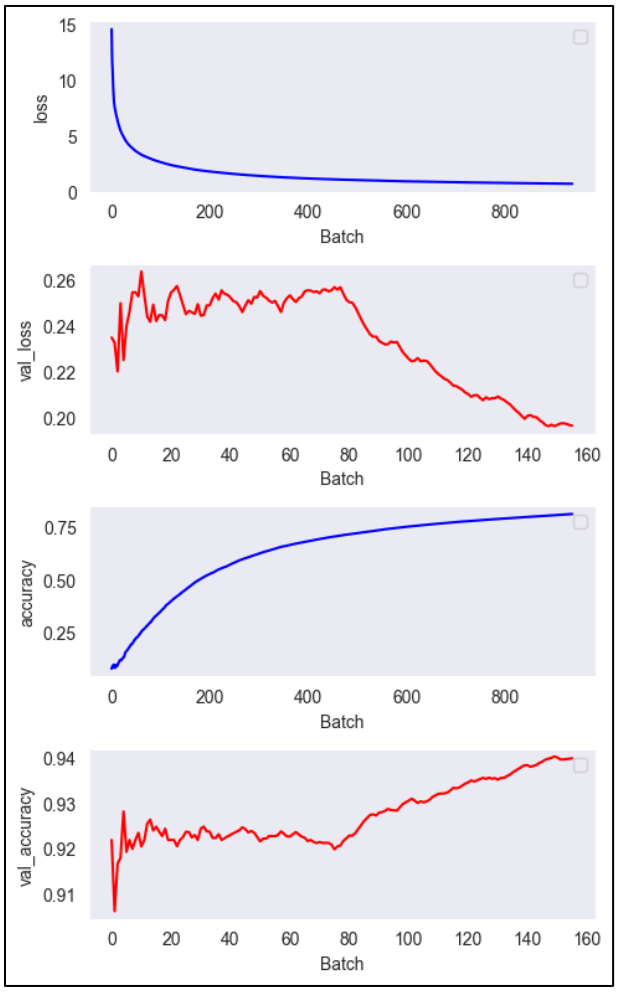


Abbildung 13: simulierter Verlauf verschiedener Metriken im Training vom Klassifikationsnetz, im 2. Durchlauf

Dieser Graph ähnelt dem ersten Graphen stark und weist eine nahezu identische Struktur auf. Es ist zu erkennen, dass dieses Netzwerk im ersten Durchlauf ebenfalls über zwei

Epochen trainiert wurde. Ähnlich wie beim simulierten SingleNet zeigt auch dieser Verlauf eine deutliche Verbesserung der Metrik zu Beginn der zweiten Epoche.

3. Graph Koordinatennetz

Der simulierte Verlauf der Trainings, vom Koordinatennetz sieht wie folgt aus:

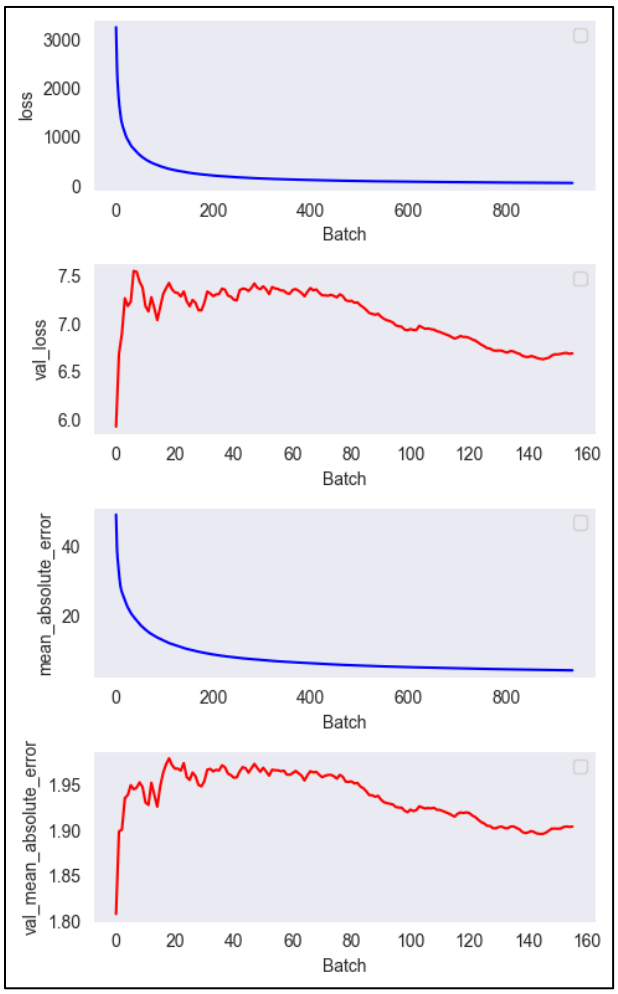


Abbildung 15: simulierter Verlauf verschiedener Metriken im Training vom Koordinatennetz, im 1. Durchlauf

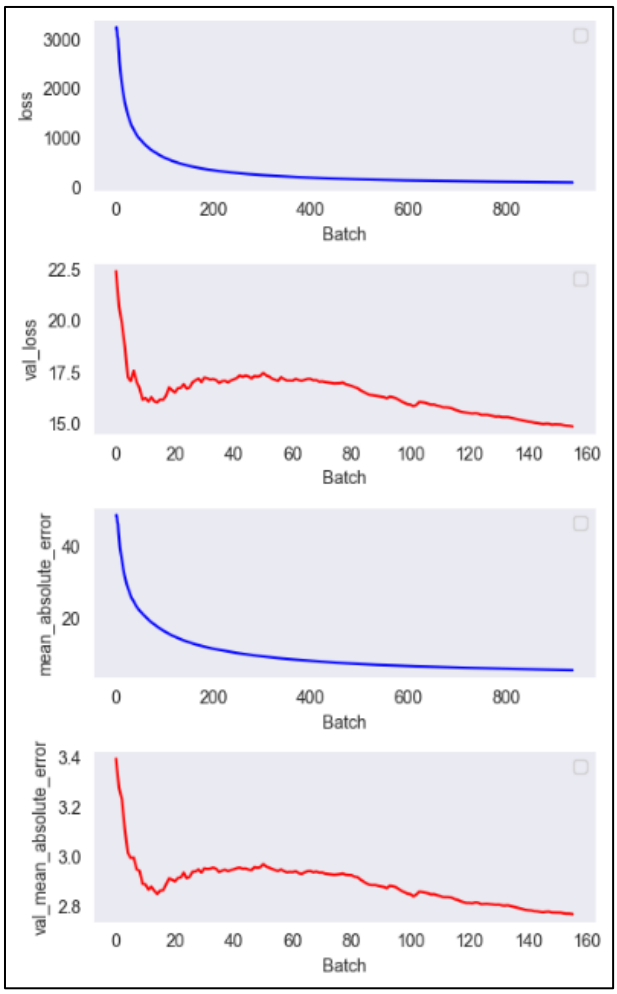


Abbildung 15: simulierter Verlauf verschiedener Metriken im Training vom Koordinatennetz, im 2. Durchlauf

Dieser Graph unterscheidet sich in seiner Struktur etwas von den Vorherigen beiden. Er zeigt zwei verschiedene Metriken und enthält keinen Epochenabschnitt. Dies liegt vor allem daran, dass das zugrunde liegende Netzwerk ein Regressionsmodell und kein Klassifikationsmodell ist. Daher wird hier die Abweichung der vorhergesagten Koordinaten gemessen, wofür der «Mean Absolute Error»“ (MAE) verwendet wird. Zudem ist erkennbar, dass dieses Modell bereits innerhalb einer einzigen Epoche vollständig trainiert wurde.

5. Interpretation der Ergebnisse

5.1 Interpretation der Trainingszeiten

Das SingleNet war im zweiten Durchlauf 0.2 Sekunden langsamer. Dies liegt daran, dass die Bayes'sche Optimierung teilweise auf zufälligen Operationen basiert, wodurch Schwankungen in den Endresultaten auftreten können.

Auffällig ist, dass im ersten Durchlauf die Trainingszeit des Koordinatennetzes höher ist als die des SingleNets. Dadurch ist bereits nach dem Training des Koordinatennetzes das DualNet langsamer als das SingleNet. Vermutlich liegt dies daran, dass zu viel Gewicht darauf gelegt wurde, das Koordinatennetz möglichst präzise zu machen, anstatt es effizienter zu gestalten. Diese Vermutung wird dadurch unterstützt, dass das Koordinatennetz im zweiten Durchlauf weniger genau, dafür aber deutlich schneller war. Daraus ergibt sich die Notwendigkeit, den optimalen Wert der Abweichung zu finden, um bewerten zu können, ob sich das DualNet gegenüber dem SingleNet tatsächlich lohnt.

Das Klassifikationsnetz zeigt mit geringen Schwankungen in beiden Durchläufen eine ähnlich schnelle Trainingszeit. Vermutlich hatte die Änderung der Abweichung hier keinen großen Einfluss auf die Trainingszeit. Dies unterstreicht die Relevanz einer gezielten Optimierung der Abweichung.

5.2 Interpretation der Netzwerkstrukturen

Es ist sehr schwierig, neuronale Netzwerkstrukturen zu interpretieren. Trotzdem kann man anhand der Ergebnisse Überlegungen anstellen, warum bestimmte Strukturen herausgekommen sind. Zum einen fällt auf, dass die beiden SingleNets eine ähnliche Verteilung von Convolutional und Dense Layern aufweisen. Das liegt wahrscheinlich daran, dass dieses Verhältnis der Layer-Anteile ungefähr optimal ist und durch den Bayes'schen (oben) Algorithmus gefunden wurde. Allerdings zeigt sich, dass das SingleNet im ersten Durchlauf deutlich weniger Parameter besitzt. Diesen Unterschied kann ich nur mit der Zufälligkeit des Bayes'schen Algorithmus erklären.

Auch auffällig ist, dass die im ersten Durchlauf bestgefundene Struktur des Klassifikationsnetzes über 300.000 Parameter hat. Dies ist vermutlich der Grund, warum die Trainingszeit im Vergleich zum zweiten Durchlauf deutlich höher war.

Die Klassifikationsnetze unterscheiden sich zwischen den Durchläufen in den Anteilen von Convolutional und Dense Layern. Dennoch haben sie eine sehr ähnliche Anzahl an Parametern und eine vergleichbare Trainingszeit, was sinnvoll erscheint.

5.3 Interpretation der simulierten Graphen

Obwohl die simulierten Graphen nicht den tatsächlichen Verlauf der bereits trainierten Modelle darstellen, zeigen sie dennoch wichtige Eigenschaften, die vermutlich auch in den echten Verläufen vorhanden waren. Wie bereits erwähnt, ist deutlich zu erkennen, dass der beste Metrik-Wert jeweils zu Beginn der zweiten Epoche erreicht wird. Dies könnte ein interessanter Ansatzpunkt für weitere Untersuchungen sein, vorausgesetzt, es handelt sich nicht um einen Messfehler.

Außerdem ist erkennbar, dass die Netzwerke ihren Zielwert meist innerhalb einer Epoche erreichen und anschließend weitertrainieren. Besonders beim Koordinatennetz fällt auf, dass der Mean Absolute Error (MAE) von Anfang an unter 5 liegt. Mit diesen Erkenntnissen könnte ein neuer Durchlauf durchgeführt werden, bei dem die Metriken genauer und kontrollierter gemessen werden. Auf diese Weise könnte das Training so gesteuert werden, dass es direkt beendet wird, sobald der Zielwert erreicht ist.

6. Diskussion

6.1 Reproduzierbarkeit der Resultate

Beim Training des neuronalen Netzwerks wurde eine GPU verwendet. Operationen, die von der GPU durchgeführt werden, sind oft nicht deterministisch, was bedeutet, dass bei gleichen Eingabedaten unterschiedliche Ergebnisse erzeugt werden. Das führt dazu, dass die Resultate nicht reproduzierbar sind.
Um dieses Problem zu umgehen, könnte der Versuch ausschließlich mit deterministischen Algorithmen durchgeführt werden. Dadurch wären die Resultate reproduzierbar, jedoch würde das Training länger dauern.

Ein Versuch von Twosigma wurde durchgeführt, bei dem ein neuronales Netzwerk zum Klassifizieren von Zahlen sowohl mit deterministischen als auch mit nicht-deterministischen Algorithmen trainiert wurde. Die Erkenntnis war, dass die Resultate reproduzierbar wurden. Es wurde jedoch auch erläutert, dass solche Algorithmen länger brauchen und es sich im maschinellen Lernen oft nicht lohnt, Trainingszeit für Reproduzierbarkeit zu opfern, vor allem, weil sich die Resultate nur gering unterscheiden, (twosigma, 2025).

6.2 Problemkomplexität

In diesem Versuch wurde zwar die Komplexität des Datensatzes erhöht, aber die Ergebnisse zeigen, dass sich die Segmentierung in diesem Fall nicht signifikant auszahlt. Es wäre spannend zu sehen, wie sich die Ergebnisse bei noch höherer Komplexität verändern, sodass eine genauere Aufteilung der Aufgaben für die Netzwerke möglich wird. Ich vermute, dass bei Problemen, bei denen das Netzwerk viel Training benötigt, um ein korrektes Ergebnis zu erzielen, wie zum Beispiel beim Unsupervised Learning, eine Aufgabenaufteilung stärker helfen könnte, da das Modell dadurch besser in die richtige Richtung geführt wird.

6.3 Datenerstellungszeit

In den Ergebnissen werden nur die endgültigen Trainingszeiten der Netzwerke berücksichtigt. Was nicht berücksichtigt wurde, ist die Optimierungszeit und die Zeit, die benötigt wurde, um den Datensatz für das Klassifikationsnetz zu modifizieren. Die Datensatzmodifikation wurde nicht berücksichtigt, weil sie sehr ineffizient implementiert wurde. Mit dieser Zeit würden die Resultate wenig aussagekräftig, da die Modifikation des Datensatzes sowie die Datensatzerstellung überwiegend viel Zeit in Anspruch nehmen. Es wäre interessant zu sehen, ob mit einer Optimierung dieses Vorgangs die benötigte Zeit minimiert werden kann. Zusätzlich wäre ein weiterer Versuch denkbar, bei dem nicht die Trainingszeit zwischen den Modellen verglichen wird, sondern die Optimierungszeit, da diese schließlich sowieso durchgeführt werden muss.

6.4 Batchgrössenoptimierung

In meinem Versuch wurde aufgrund von Hardware-Limitationen eine fixe Batchgröße (oben) von 64 verwendet, was die Größe des Netzwerks einschränkt. Eine mögliche Weiterentwicklung wäre, mit besserer Hardware die Batchgröße im Optimierungsprozess als Parameter festzulegen. Dadurch könnte eine größere Auswahl an Netzwerkstrukturen berücksichtigt werden, was zu aussagekräftigeren Ergebnissen führen würde.

6.5 Optimierung der Pixelabweichung

Wie bereits erwähnt, zeigen die Ergebnisse der Durchläufe, dass der optimale Zielwert für das Koordinatenmodell ermittelt werden muss. Um dies zu erreichen, müsste das Problem mit der Datenerstellungszeit (oben) gelöst werden, sodass das gesamte Training des DualNet – also sowohl das Koordinatennetz als auch das Klassifikationsnetz – in einem einzigen Optimierungsschritt durchgeführt werden kann. Durch diese Integration könnte die Abweichung als Optimierungsparameter angegeben werden, sodass sie nicht mehr manuell festgelegt werden muss. Stattdessen würde der Optimierungsalgorithmus den besten Wert automatisch finden.

7. Zusammenfassung

Die Segmentierung eines neuronalen Netzwerks wurde in diesem Versuch getestet, jedoch zeigte sich dabei keine signifikante Verbesserung der Leistung. Das segmentierte Modell war in den durchgeführten Messungen lediglich maximal 0.2 Sekunden schneller als das nicht-segmentierte Modell. Dieser Unterschied ist zu gering, um von einer klaren Überlegenheit des segmentierten Ansatzes sprechen zu können.

Trotzdem liefert der Versuch Hinweise darauf, dass dieser Ansatz Potenzial hat. Insbesondere könnte die Segmentierung bei einer optimierten Umsetzung und einem komplexeren Datensatz durchaus zu besseren Ergebnissen führen. Eine entscheidende Rolle spielen hierbei vor allem ein verbessertes Optimierungsverfahren und ein geeigneteres Training der Netzstruktur.

Aufgrund fehlenden Vorwissens sowie begrenzter Zeit und Ressourcen war es in diesem Rahmen jedoch nicht möglich, diese Aspekte vollständig auszuschöpfen. Künftige Arbeiten könnten an diesen Punkten ansetzen, um das Potenzial segmentierter neuronaler Netzwerke unter verbesserten Bedingungen genauer zu untersuchen und möglicherweise deutliche Vorteile gegenüber nicht-segmentierten Modellen nachzuweisen.

Quellen

geeksforgeeks. (02. 01 2025). Von https://www.geeksforgeeks.org/what-is-a-batch-in-tensorflow/ abgerufen

Keras API. (9. Dezember 2024). Von https://keras.io/api/layers/regularization_layers/dropout/ abgerufen

Keras API. (9. Dezember 2024). Von https://keras.io/api/layers/convolution_layers/convolution2d/ abgerufen

Keras API. (9. Dezember 2024). Von https://keras.io/api/layers/core_layers/dense/ abgerufen

saturncloud. (9. Dezebmer 2024). Von https://saturncloud.io/blog/understanding-the-difference-between-flatten-and-globalaveragepooling2d-in-keras/ abgerufen

Tensorflow APi. (9. Dezember 2024). Von https://www.tensorflow.org/api_docs/python/tf/keras/layers/Conv2D abgerufen

twosigma. (02. 01 2025). Von https://www.twosigma.com/articles/a-workaround-for-non-determinism-in-tensorflow/ abgerufen

Wikipedia. (19. 12 2024). Von https://de.wikipedia.org/wiki/Bayes%E2%80%99sche_Optimierung abgerufen

Bei dieser Arbeit wurde ChatGPT als Hilfsmittel verwendet, um Textabschnitte zu korrigieren und umzuschreiben.

Der gesamte Code für diese Arbeit, einschließlich aller Trainingsergebnisse und Grafiken, ist über folgenden Link verfügbar: https://github.com/alessio-maffioletti/matura/tree/main

Eigenständigkeitserklärung

Hiermit bestätige ich, dass ich meine Maturitätsarbeit selbstständig und nur unter Zuhilfenahme der in den Verzeichnissen oder in den Anmerkungen genannten Quellen und KI-/ LLM-Tools angefertigt habe. Die Mitwirkung von anderen Personen hat sich auf Beratung und Korrekturlesen beschränkt. Alle verwendeten Unterlagen und Gewährspersonen sind vollständig aufgeführt». Es folgen Ort, Datum und Unterschrift.