CIFAR10 mit PyTorch klassifizieren

Neuronale Netze in der Bildverarbeitung

Jan Hoegen Nico Weber

28. Oktober 2025

Hochschule Karlsruhe
University of Applied Sciences

Inhaltsverzeichnis

Allgemeines zu PyTorch

PyTorch im Überblick

Was ist PyTorch?

- Python-Bibliothek für Deep Learning
- Stark verbreitet in Forschung und Lehre [1]
- Unterstützt dynamische Berechnungsgraphen („Define-by-Run“)

Warum PyTorch?

- Einfache und flexible Modellimplementierung
- Direkte Nutzung von GPU-Beschleunigung
- Große Community, viele Tutorials und Ressourcen

Grundlegende Konzepte von Pytorch

Autograd Berechnet Gradienten automatisch für Backpropagation

`nn.Module` Basis für selbstdefinierte Modelle, enthält vordefinierte Layers

DataLoader Einfaches Laden, Batchen und Parallelisieren von Datensätzen

Optimizer Vorgefertigte Optimierer, z.B. **`Adam`**

Eigenes Modell zu CIFAR 10

Aufgabenstellung

- **Aufgabe:** Klassifikation von CIFAR-10 Bildern
- **Datensatz:**
  - 10 Klassen
  - 60.000 Bilder, Größe $32 \times 32 \times 3$
  - Trainingsset: 49.000 Bilder
  - Validierungsset: 1.000 Bilder
  - Testset: 10.000 Bilder
- **Ziel:** Modell in *maximal 10 Epochen* trainieren, um Bilder korrekt zu klassifizieren

Architektur des Modells

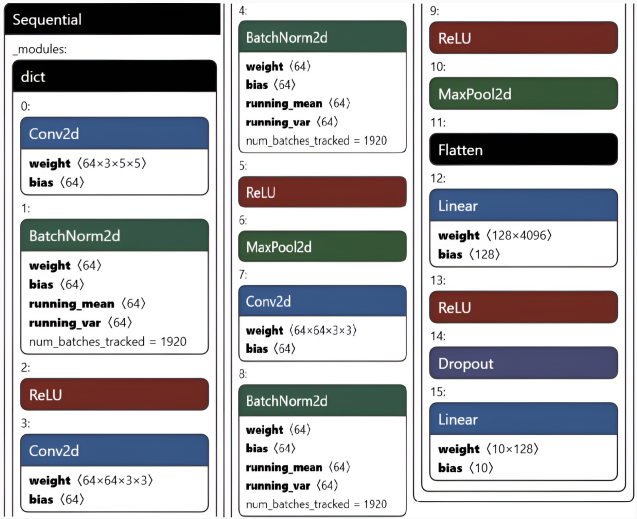


Abbildung 1: Architektur des eigenen CNN-Modells.

Ergebnisse des besten Modells

- Beste Hyperparameter:
  - ch1 = 64, ch2 = 64, ch3 = 64
  - Lernrate = 8.6e-4
  - Weight Decay = 3.81e-6
- Best Validation Accuracy: 75.70%
- Trainings-Accuracy: 73%
- Test Accuracy: 75.05%

Accuracy für verschiedene Klassen

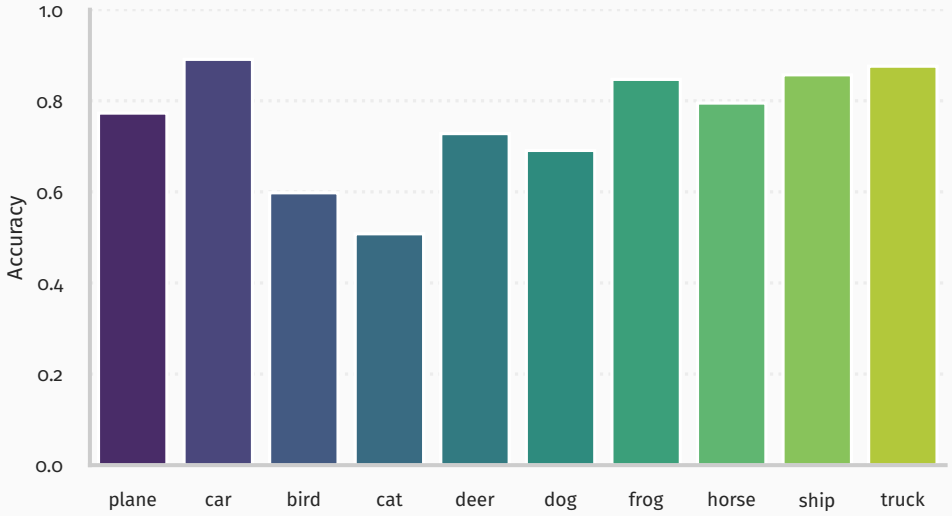


Abbildung 2: Test Accuracy für verschiedene Klassen.

Confusion Matrix

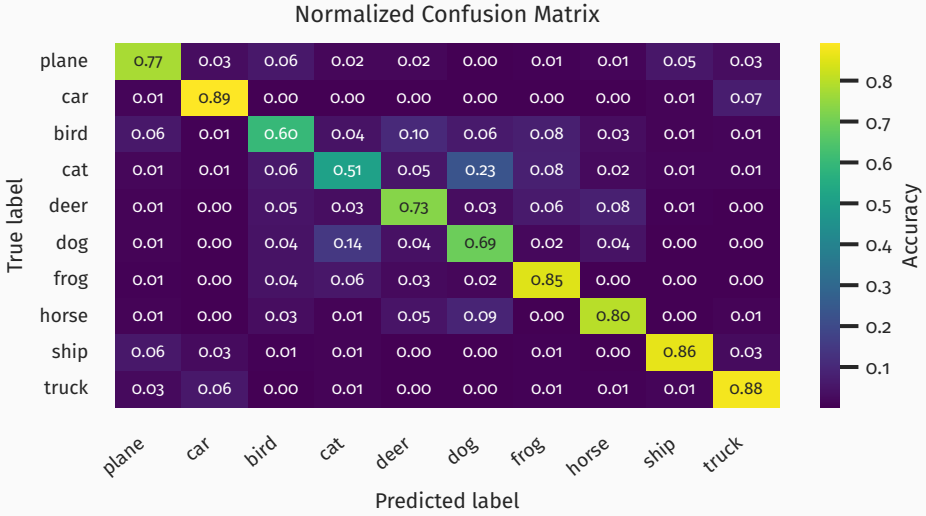


Abbildung 3: Normierte Confusion Matrix für das Testset.

Beispielvorhersagen

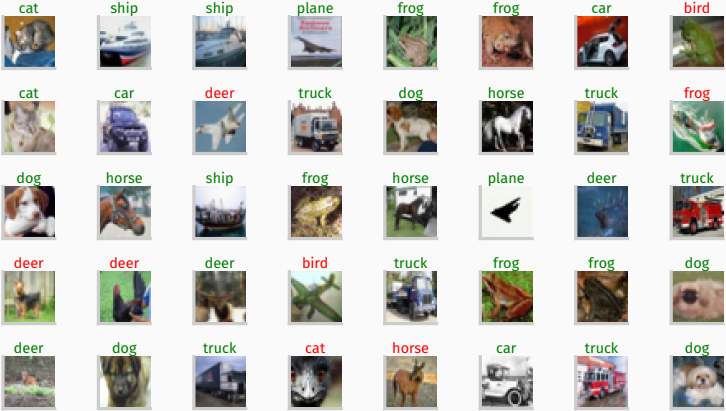


Abbildung 4: Beispielhafte Vorhersagen des Modells.

Legende: Korrekte Klasse: grün. Falsche Klasse: rot.

Training und Hyperparameter

Einstellungen für das Training

Trainingsdaten:

- 49.000 Trainingsbilder, Größe $32 \times 32 \times 3$
- Batchgröße: 256
- DataLoader-Worker: 12 (hohe Parallelisierung)

Training:

- Trainierbare Parameter: 604 810
- Iterationen pro Epoche: $\lceil 49{,}000/256 \rceil = 192$
- Epochen je Hyperparameter-Kombination: 10

Hyperparameter-Optimierung:

- Bayesian Search mit 20 Hyperparameter-Kombinationen (Trials)
- Optimierte Parameter: ch1, ch2, ch3, Lernrate, Weight Decay

Genauigkeit während des Trainings

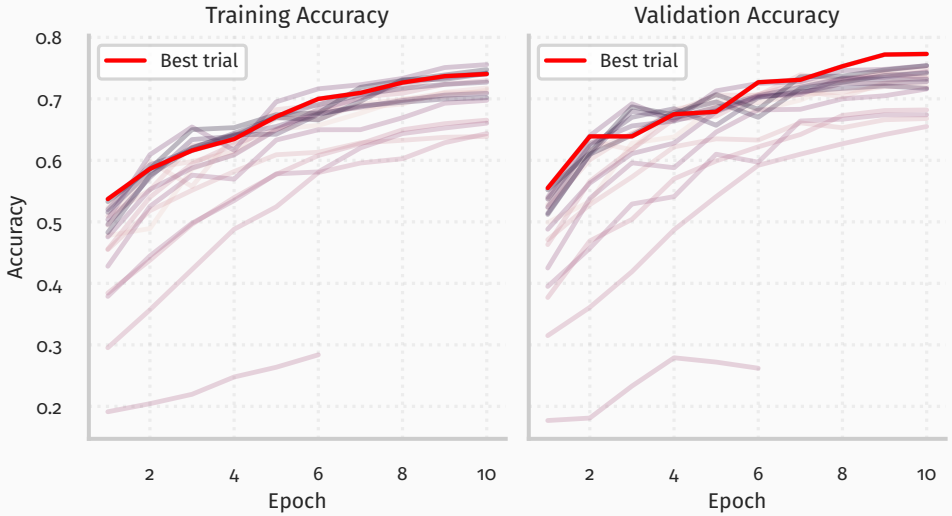


Abbildung 5: Validation und Training für verschiedene Traials.

Early Stopping

Motivation:

- Verhindert Overfitting
- Stoppt das Training, wenn die Validierungsgenauigkeit über mehrere Epochen nicht steigt

Prinzip:

- Lege Grenzwert `patience` fest
- Nach jeder Epoche: Bestwert der Validierungsgenauigkeit speichern
- Zähle Epochen ohne Verbesserung: `epochs_no_improvement`
- Bei neuer Verbesserung: Zähler `epochs_no_improvement` zurücksetzen
- Stoppe Training, sobald `epochs_no_improvement` $\geq$ `patience`

Hardware-Auslastung

Hardware:

- GPU: RTX 4060 Ti, 16 GB VRAM
- CPU: Ryzen 5 7600X, 12 Kerne
- RAM: 32 GB DDR5

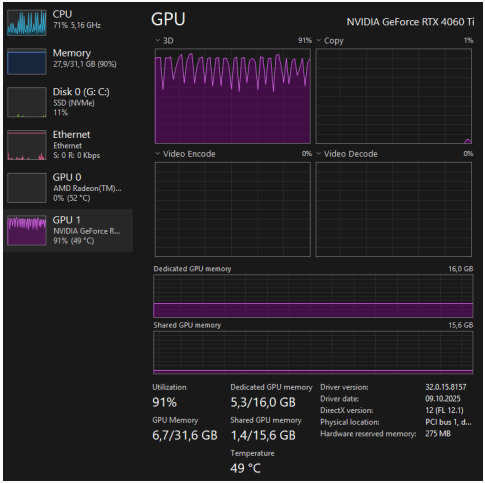


Abbildung 6: Auslastung gemäß Task Manager

Laufzeiten pro Trial

Gesamtdauer: ca. 15 Minuten

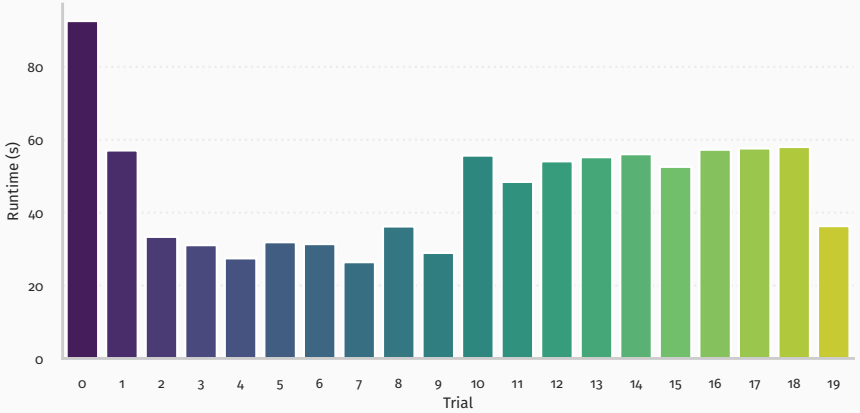


Abbildung 7: Laufzeit des Trainings je Trial.

Parametertuning mit Bayesian Search

Bayesian Search Erklärt

- Bayesian Search ist eine intelligente Methode zur Optimierung von Hyperparametern.
- Ziel: Maximierung der Validierungsgenauigkeit
  $f(\text{Parameter}) \rightarrow \text{Validation Accuracy}$.
- Idee:
  - Es wird ein Modell erstellt, das die unbekannte Zielfunktion f abschätzt.
  - Nach jeder Evaluierung wird dieses Modell mit den neuen Ergebnissen aktualisiert.
  - Eine *Acquisition Function* wählt die nächsten Parameterwerte aus – als Kompromiss zwischen **Exploration** (neue Bereiche testen) und **Exploitation** (bestehende gute Bereiche verfeinern).
- Vorteil: Findet gute Parameter mit deutlich weniger Versuchen als Random oder Grid Search.

Animation zu Bayesian Search

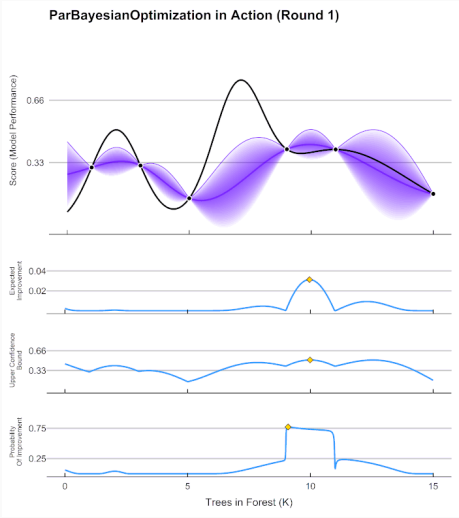


Abbildung 8: Bayes'sche Optimierung eines Scores für einen Random-Forest-Klassifizierer

Quelle: [2]
Legende: *x-Achse:* Parameter des Random-Forest-Klassifizierers. *Schwarz:* Zielfunktion. *Lila:* Modellierte Funktion mit Unsicherheitsbereich ±1 Standardabweichung. *Expected Improvement:* Erwarteter Zugewinn gegenüber dem aktuellen Bestwert. *Upper Confidence Bound:* Suche vielversprechende, aber unerkundete Bereiche. *Probability of Improvement:* Wahrscheinlichkeit, dass ein neuer Punkt besser ist als der bisherige Bestwert.

Anwendung auf unser Modell

- Anstatt zufällig Parameterkombinationen zu testen (Random Search) oder alle möglichen Kombinationen (Grid Search), wurde **Bayesian Search** verwendet.
- Das Modell der Funktion Validation Accuracy $=$ f(Hyperparameter) wird kontinuierlich angepasst.
- Neue Vorschläge für Hyperparameter werden auf Basis bisheriger Ergebnisse erzeugt.

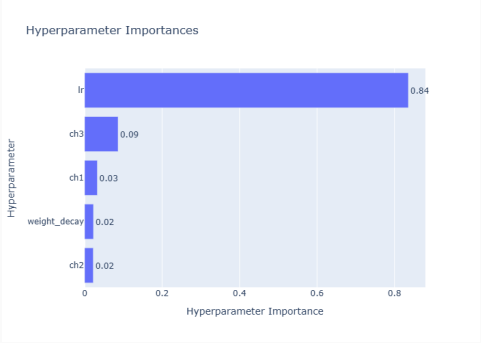


Abbildung 9: Relative Wichtigkeit der Hyperparameter

Bayesian Search Ergebnisse

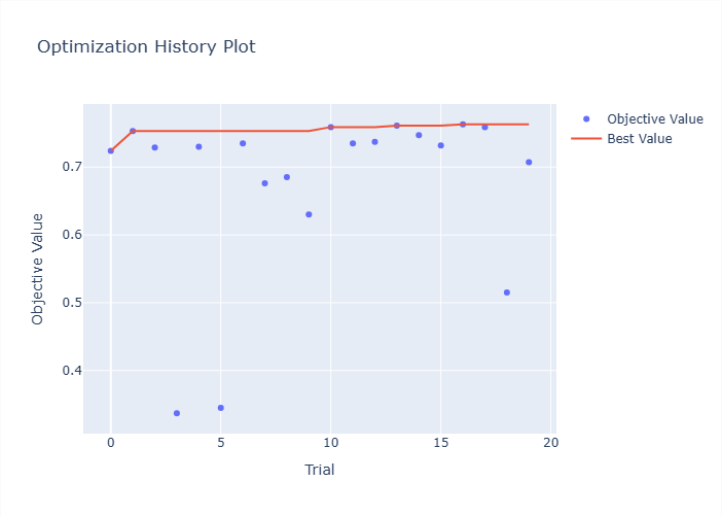


Abbildung 10: Verlauf der Validierungsgenauigkeit über die Trials (je höher, desto besser).

Ausblick

Ausblick

- **Vergleich der Optimierungsmethoden:** Quantitativer Vergleich von Bayesian Search mit Random Search und Grid Search
- **Alternative Modelle:** Testen anderer Netzwerkarchitekturen (z.B. mehr convolution) auf CIFAR-10
- **Hyperparameter-Erweiterungen:** Batchgröße, Dropout-Rate oder Lernrate als zusätzliche Parameter variieren

Fragen?

Literatur

[1] J. Bauer, *ComputerVision2: Neuronale Netze in der Bildverarbeitung*. 18. Juni 2025.

[2] AnotherSamWilson, **„Bayesian optimization of a function with a Gaussian process“,** besucht am 27. Okt. 2025. Adresse: https://github.com/AnotherSamWilson/ParBayesianOptimization

[3] S. Subramanian, S. Juarez, C. Breviu, D. Soshnikov und A. Bornstein, **„PyTorch Tutorials: Beginner Basics“,** besucht am 26. Okt. 2025. Adresse: https://docs.pytorch.org/tutorials/beginner/basics/intro.html

Wichtige Konzepte im Training i

Regularisierung:

- **Dropout:** Deaktiviert zufällig Neuronen während des Trainings, um Overfitting zu reduzieren und die Generalisierung zu verbessern.
- **Weight Decay:** Fügt einen Strafterm für große Gewichte hinzu und verhindert dadurch übermäßig komplexe Modelle.

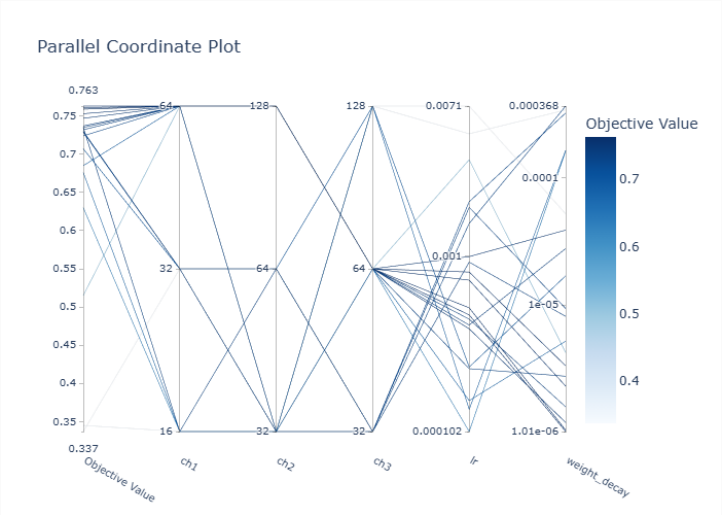
Feature-Reduktion:

- **MaxPooling:** Verringert die räumliche Auflösung von Feature-Maps, indem pro Bereich nur der größte Aktivierungswert beibehalten wird. Reduziert Rechenaufwand und sorgt für Translationstoleranz (robuster gegenüber kleinen Verschiebungen im Bild).

Wichtige Konzepte im Training ii

Optimierung:

- **Adam:** Optimierer, der Momentum und RMSProp kombiniert. Passt die Lernrate adaptiv für jedes Gewicht an.
- **Learning Rate:** Schrittweite, mit der die Gewichte in Richtung des Gradienten aktualisiert werden. Zu groß → instabil, zu klein → langsames Lernen.


Parallel Coordinate Plot
0.763
0.75
0.7
0.65
0.6
0.55
0.5
0.45
0.4
0.35
0.337
Objective Value
64
32
16
ch1
128
64
32
ch2
128
64
32
ch3
0.0071
0.001
0.000102
lr
0.000368
0.0001
1e-05
1.01e-06
weight_decay
Objective Value
0.7
0.6
0.5
0.4