



DEEP LEARNING REPORT

Hand Gesture Recognition

durchgeführt am Studiengang
Informationstechnik und System-Management
Fachhochschule Salzburg GmbH

vorgelegt von
Justin Michael Schönberg
Florian Hinterberger
Patrick Hofmann

Studiengangsleiter: FH-Prof. DI Dr. Gerhard Jöchtl
Betreuer: FH-Prof. DI (FH) Dr. Werner Kaltner-Pomwenger, MSc

Puch/Salzburg, Dezember 2024

Inhaltsverzeichnis

1 Einleitung	1
2 Datensatz	2
3 Vorverarbeitung	4
4 Implementierung eines Convolutional Neural Networks (CNN)	5
4.1 Training	5
4.2 Evaluierung	7
5 Training	8
5.1 Trainings-Log auf dem Watson-Cluster	10
6 Evaluierung	11
6.1 Analyse der Loss-Kurven	11
6.2 Klassifikationsberichte und Performance-Metriken	12
6.3 Confusion-Matrizen	13
6.4 Vergleich mit HAGRID-Baseline-Modellen	14
6.5 Zusammenfassung	14
7 Demo	15
8 Fazit	16
Literaturverzeichnis	17

1 Einleitung

In der heutigen Zeit spielt die Interaktion zwischen Mensch und Maschine eine immer größere Rolle. Ein besonders innovativer Ansatz besteht darin, Computer in die Lage zu versetzen, menschliche Handgesten zu erkennen und zu interpretieren. Dies ermöglicht eine intuitive Steuerung von Geräten, die weit über klassische Eingabemethoden hinausgeht. Handgestenerkennung findet Anwendungen in verschiedensten Bereichen wie der virtuellen Realität, Gebärdensprachübersetzung, robotergestützter Steuerung oder auch im Bereich der Barrierefreiheit.

Im Rahmen dieses Projekts wird ein Vergleich verschiedener Deep-Learning-Modelle durchgeführt, wobei der Fokus auf der Evaluation von ResNet-Architekturen liegt. Als Grundlage dient der HAGRID-Datensatz. Ziel ist es, die Leistungsfähigkeit eines vortrainierten ResNet-Modells in zwei unterschiedlichen Setups zu untersuchen:

- Im ersten Setup wird das vortrainierte Netz mithilfe von Transfer Learning eingesetzt, wobei alle Layer bis auf den abschließenden Klassifikations-Layer eingefroren sind.
- Im zweiten Setup wird dasselbe vortrainierte Netz verwendet, jedoch werden alle Layer freigegeben und das gesamte Modell vollständig nachtrainiert.

Anschließend werden die beiden Ansätze hinsichtlich ihrer Performance-Metriken und der benötigten Trainingszeit verglichen, um abzuwegen, inwieweit es sinnvoll ist, ein gesamtes Netz nachzutrainieren.

Dieses Projekt bietet die Möglichkeit, theoretische Konzepte aus dem Bereich des maschinellen Lernens praxisnah umzusetzen und tiefere Kenntnisse über die Anwendung von Deep-Learning-Methoden zu gewinnen. Insbesondere die Arbeit mit einem spezialisierten Datensatz wie HAGRID stellt interessante Herausforderungen bei der Modellierung, Vorverarbeitung und Optimierung dar.

2 Datensatz

Der HaGRIDv2-Datensatz umfasst 1.086.158 FullHD RGB Bilder von 65.977 Personen, die jeweils mit mindestens einer eindeutigen Szene verbunden sind. Die Probanden sind Erwachsene im Alter von 18 Jahren und älter. Die meisten Bilder wurden in Innenräumen aufgenommen, was erhebliche Unterschiede bei den Lichtverhältnissen, sowohl bei künstlichem als auch bei natürlichem Licht, erkennen lässt. Darüber hinaus enthält der Datensatz schwierige Szenarien, wie z. B. Personen, die vor oder hinter einem Fenster stehen. Die Gesten wurden in Entferungen von **0,5 bis 4 Metern** von der Kamera demonstriert, um unterschiedliche Perspektiven und Umgebungen zu gewährleisten[1].

Der Datensatz umfasst insgesamt 33 Klassen, wobei eine dieser Klassen die Kategorie "no_gesture" repräsentiert. Eine Übersicht über die Gestenklassen ist in Abbildung 1 dargestellt.

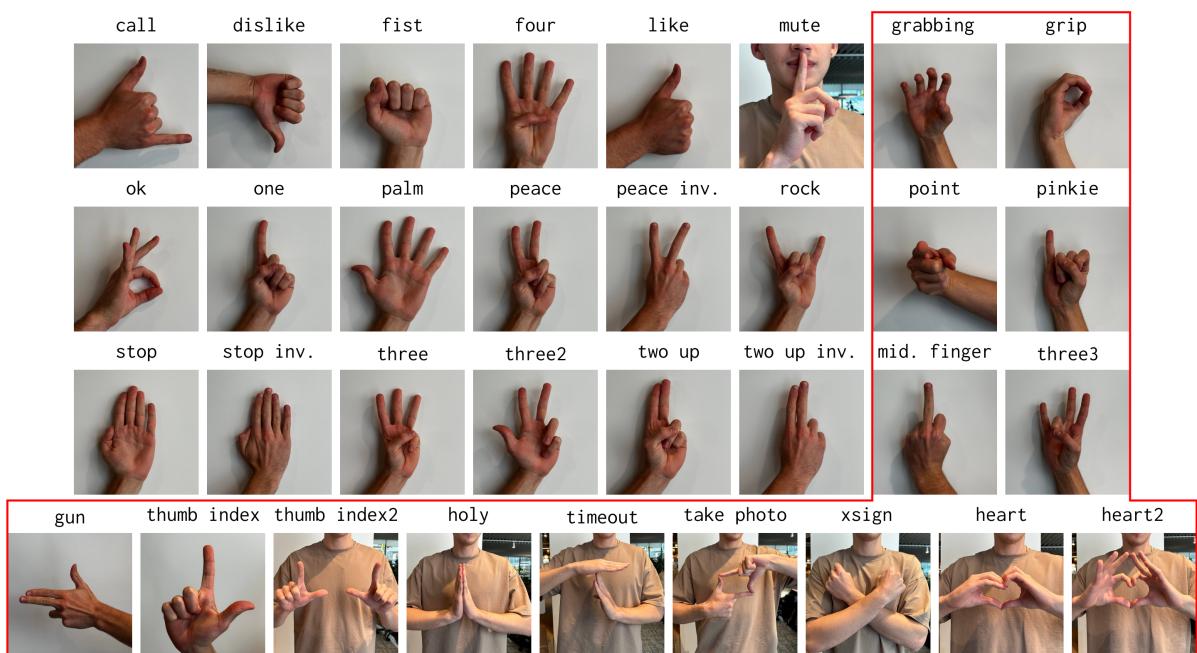


Abbildung 1: Alle 33 Handgesten-Klassen nach HAGRID [1]

Die Annotation des HAGRID-Datensatzes wird in Form einer JSON-Datei gespeichert und enthält detaillierte Informationen zu jedem Bild sowie zu den darin enthaltenen Handgesten. Für unser Projekt werden nur ein Teil der zur Verfügung gestellten Informationen benötigt:

- **image_id**: Zuordnung zu den Bildern des Datensatzes.
- **bboxes**: Array von Bounding Boxes, das die Positionen und Größen der erkannten Handgesten im Bild angibt. Diese wird im COCO-Format [X (oben links), Y (oben links), Breite, Höhe] angegeben.
- **labels**: Array, das die Labels der erkannten Handgesten enthält.

In Quelltext 1 ist ein Beispielausschnitt der Annotation zu sehen. Dieser enthält zwei Handgesten, `three3` und `no_gesture`, die durch die zugehörigen Bounding Boxes eindeutig identifizierbar sind.

```
1 ...
2 "136ad98f-5341-4f4d-a779-e86f802cbac2": {
3     "bboxes": [
4         [
5             0.3350748,
6             0.33349978,
7             0.05312771,
8             0.05859395
9         ],
10        [
11            0.45263392,
12            0.35947449,
13            0.05834404,
14            0.0528153
15        ]
16    ],
17    "labels": [
18        "three3",
19        "no_gesture"
20    ],
21    ...
22 },
23 ...
```

Quelltext 1: HAGRID Annotation Ausschnitt

3 Vorverarbeitung

Für die Vorverarbeitung des Datensatzes ist es von zentraler Bedeutung, dass nur die ausgeschnittenen Hände extrahiert werden, um das Modell gezielt auf diese Bereiche zu trainieren. Man arbeitete hier nach dem Pytorch Dataset, Dataloader und Transform [2] Ansatz und entwickelte basierend auf dem von HAGRID vorgegebenen Dataset eine eigene Dataset Klasse **BBoxClassificationDataset**, die es ermöglicht, Bilder und deren Label zu extrahieren. Anschließend wird mittels der Transformationsfunktion **CropToB-Box** der relevante Bildausschnitt, der die Hand darstellt, extrahiert. Diese Vorgehensweise stellt sicher, dass nur die für das Modell relevanten Bildregionen verwendet werden.

Der nachfolgende Abschnitt veranschaulicht diesen Ablauf.

1. Laden des Originalbilds

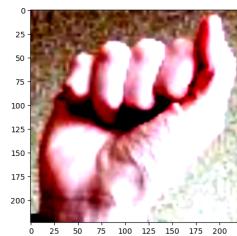


2. Laden der Annotation

boxes: [89.8317, 274.4490, 160.8141, 354.6785])

label: fist

3. Transform anwenden



Der Transform wendet auch die ImageNet Normalisierung und Skalierung an, um die korrekte Trainingsbasis für Modelle wie ResNets zu liefern.

Nun konnten mit einem Dataloader ausgeschnittene Handbilder und deren zugewiesene Klasse ausgelesen werden. Damit liegen die Trainingsdaten passend für die Netze vor.

4 Implementierung eines Convolutional Neural Networks (CNN)

In diesem Projekt wurde ein Convolutional Neural Network (CNN) zur Klassifikation von Handgesten implementiert und trainiert. Als Framework wurde FastAI genutzt, eine leistungsstarke Deep-Learning-Bibliothek, die sowohl modulare und flexible Komponenten für Forschungsanwendungen als auch benutzerfreundliche Werkzeuge für die schnelle Entwicklung von Modellen bietet. Dank der engen Integration mit PyTorch und der schichtweisen Architektur bietet FastAI eine klare Abstraktion häufig verwendeter Deep-Learning-Muster und kombiniert Benutzerfreundlichkeit mit hoher Leistung[3].

4.1 Training

Das Training des CNN erfolgte unter Verwendung der `Learner`-Funktion¹ aus der FastAI-Bibliothek. Der `Learner` bündelt alle wesentlichen Komponenten des Modells, einschließlich der Modellarchitektur, der Datenquellen und der Optimierungsmethoden. Die Trainingsroutine basiert auf der Methode `learn.fit_one_cycle(epochs, learning_rate)`, bei der die Anzahl der Trainings-Epochen und die Lernrate definiert werden.

Für die Initialisierung des `Learner` wurden folgende Komponenten spezifiziert:

- **Zielklassen:** Die Anzahl der Klassen wurde aus den Konfigurationsdateien (HAGRID-constants) abgeleitet.
- **Modellarchitektur:** Das Modell wurde in Abhängigkeit der gewählten Setup-Parameter wie folgt konfiguriert:
 1. Import eines vortrainierten Modells (z. B. ResNet) mit den entsprechenden vortrainierten Gewichten.
 2. Modifikation des Fully-Connected Layers, um die Anzahl der Ausgabeeinheiten an die Anzahl der Zielklassen anzupassen.
 3. Optionales Laden eines bereits trainierten Modells zur weiteren Feinabstimmung.
 4. Im `frozen`-Setup wurden alle Schichten außer dem Fully-Connected Layer eingefroren, um die Generalisierungsfähigkeit des Modells zu fördern.

¹ <https://docs.fast.ai/learner.html#learner>

- **Dataloader:** Basierend auf den in Kapitel 3 beschriebenen Vorverarbeitungsmethoden wurden Trainings- und Validierungs-Datenlader erstellt.

Die Implementierung des **Learner** ist in Quelltext 2 dargestellt. Als Verlustfunktion wurde **CrossEntropyLossFlat** eingesetzt, eine speziell für Klassifikationsprobleme optimierte Variante der Cross-Entropy-Verlustfunktion. Darüber hinaus wurden verschiedene *Callback*-Funktionen spezifiziert, darunter:

- **ProgressCallback:** Visualisierung des Trainingsfortschritts in Echtzeit.
- **CSVLogger:** Speicherung der Trainingsmetriken in einer Log-Datei.
- **EarlyStoppingCallback:** Automatischer Abbruch des Trainings bei fehlender Verbesserung des Validierungsverlusts.
- **SaveModelCallback:** Speichern des Modells mit der besten Validierungsleistung.

```

1 ...
2 num_classes = len(constants.targets)
3 model = get_model(model_type, setup, num_classes)
4 transforms = Transforms.CropToBBox()
5 dataloaders = get_dataloaders(conf, batch_size, transforms)
6 train_dl, val_dl = dataloaders
7
8 logger.info("Starting training process...")
9 learn = Learner(
10     dls=DataLoaders(train_dl, val_dl),
11     model=model,
12     loss_func=CrossEntropyLossFlat(),
13     metrics=[accuracy],
14     cbs=[
15         ProgressCallback(),
16         CSVLogger(fname=stats_dir / f"{model_type}_{setup}_training_logs.csv",
17                   append=True),
18         EarlyStoppingCallback(monitor='valid_loss', patience=patience),
19         SaveModelCallback(monitor='valid_loss', fname=model_dir / f"{model_type}_{setup}_best")
20     ]
21 )
22 ...

```

Quelltext 2: Implementierung des Learners

4.2 Evaluierung

Die Evaluierung des trainierten Modells erfolgte mittels der Funktion `evaluate_model`, die darauf abzielt, die Leistungsfähigkeit des Modells auf einem separaten Testdatensatz zu bewerten und umfassende Metriken zur Klassifikationsqualität bereitzustellen. Die Evaluierungsroutine umfasst die folgenden Schritte:

- **Erstellung des Testdatensatzes:** Der Testdatensatz wurde durch die Anwendung identischer Vorverarbeitungsmethoden wie im Trainingsprozess erstellt.
- **Vorhersagen generieren:** Die Methode `learn.get_preds` wurde verwendet, um die Modellvorhersagen auf dem Testdatensatz zu berechnen. Die vorhergesagten Klassen (`y_pred`) wurden als Indizes der höchsten Wahrscheinlichkeiten aus den Modellvorhersagen (`preds`) ermittelt, während die tatsächlichen Klassen (`y_true`) aus den Zielwerten des Testdatensatzes extrahiert wurden.
- **Klassifikationsbericht:** Ein umfassender Klassifikationsbericht wurde mithilfe der Funktion `classification_report` erstellt. Dieser enthält Kennzahlen wie Präzision, Recall und F1-Score für jede Klasse. Der Bericht wurde zur weiteren Analyse in einer Textdatei gespeichert.
- **Konfusionsmatrix:** Zur Visualisierung der Klassifikationsleistung wurde eine Konfusionsmatrix generiert, die die Häufigkeiten der tatsächlichen und vorhergesagten Klassen darstellt.

5 Training

Das Training des Convolutional Neural Network (CNN) wurde auf dem leistungsstarken Watson-Cluster der Fachhochschule Salzburg durchgeführt. Dabei kam der Knoten C1 zum Einsatz, der über eine High-Performance-Hardwarekonfiguration verfügt. Dieser Knoten ist mit acht NVIDIA GeForce RTX 2080Ti GPUs, 88 GB GPU-Speicher, 20 CPU-Kernen und 230 GB Arbeitsspeicher ausgestattet, was eine effiziente Durchführung rechenintensiver Trainingsprozesse ermöglicht.

Für das Ressourcen- und Job-Management wurde der SLURM-Cluster-Manager eingesetzt. Der Trainingsprozess wurde durch das Starten eines SLURM-Jobs mittels des Befehls

`sbatch train_resnet34_frozen.sh` initiiert. Das zugehörige Shell-Skript, dargestellt in Quelltext 3, aktiviert zunächst eine virtuelle Umgebung mit Conda, die sämtliche benötigten Abhängigkeiten enthält. Anschließend wird das Python-Skript `TrainClassifier.py` ausgeführt, um den Trainingsprozess zu starten.

Das trainierte Modell, `ResNet34`, wurde im `frozen`-Modus verwendet, wobei alle vortrainierten Schichten bis auf den Fully-Connected-Layer eingefroren wurden. Der Fully-Connected-Layer wurde an die spezifische Problemstellung der Handgestenerkennung angepasst. Der Trainingsdatensatz bestand aus einer Teilmenge von 10.000 Bildern, und das Modell wurde über 3 Epochen mit einer Batch-Größe von 64 und einer Lernrate von 0.001 trainiert. Der Fortschritt des Trainings sowie potenzielle Fehler wurden detailliert in spezifischen Log-Dateien dokumentiert.

Die Implementierung ist flexibel gestaltet und erlaubt eine einfache Anpassung des Trainings durch Modifikation der Parameter im Shell-Skript. Es können diverse Einstellungen wie das Modell (`-model`), die Trainingskonfiguration (`-setup`), die Anzahl der Trainingsdaten (`-subset`), die Anzahl der Epochen (`-epochs`), die Batch-Größe (`-batch_size`), die Lernrate (`-learning_rate`) und Early Stopping (`-patience`) individuell konfiguriert werden. Diese Struktur ermöglicht eine effiziente Durchführung von Experimenten mit unterschiedlichen Modellen und Trainingsparametern.

```
1 #!/bin/bash
2 #
3 #SBATCH --job-name=train_hagrid    # Job-Name
4 #SBATCH --output=Log/%x_%j.out      # Ausgabe-Datei (enthält Job-Name und
5 #           ID)
6 #SBATCH --error=Log/%x_%j.err       # Fehler-Datei (enthält Job-Name und ID
7 #           )
8 #SBATCH --gres=gpu:2                # Anzahl der GPUs
9 #SBATCH --ntasks=1                  # Anzahl der Prozesse
10 #SBATCH --time=5-00:00              # Laufzeit im Format D-HH:MM
11 #SBATCH --mem-per-cpu=4G            # Speicher pro CPU
12 #SBATCH --cpus-per-task=8           # CPU-Kerne pro Aufgabe
13
14
15 eval "$(conda shell.bash_hook)"
16 conda activate /home2/phofmann/miniconda/envs/hagrid/
17
18 cd /srv/GadM/Datasets/Tmp/Hand-Gesture-Recognition/2_Modelling/
19 python TrainClassifier.py --model=resnet34 --setup=frozen --subset=10000
20     --epochs=3 --batch_size=64 --learning_rate=0.001 --patience=1
21
22 exit
```

Quelltext 3: SLURM-Skript zur Trainingsinitialisierung

5.1 Trainings-Log auf dem Watson-Cluster

Während des Trainingsprozesses wurden detaillierte Logs erstellt, um den Fortschritt sowie die Ergebnisse jeder Epoche zu dokumentieren. Diese Logs umfassen zentrale Informationen wie den Trainings- und Validierungsverlust, die Genauigkeit, die Laufzeit pro Epoche sowie Hinweise auf potenzielle Verbesserungen des Modells. Ein Beispielauszug aus den Log-Dateien des Trainings eines ResNet34-Modells im frozen-Setup ist in Abbildung 2 dargestellt.

```
(base) fhinterberger@interface:/srv/GadM/Datasets/Tmp/Hand-Gesture-Recognition$ cat Log/train_hagrid_20586.out
2024-11-14 12:17:18,518 - __main__ - INFO - Initializing model: resnet34 with frozen setup...
2024-11-14 12:17:19,121 - __main__ - INFO - Loading existing model...
2024-11-14 12:17:19,354 - __main__ - INFO - Model initialized successfully.
2024-11-14 12:17:19,354 - __main__ - INFO - Loading datasets...
2024-11-14 12:18:35,291 - __main__ - INFO - Datasets loaded successfully.
2024-11-14 12:18:35,292 - __main__ - INFO - Starting training process...
2024-11-14 12:18:36,853 - __main__ - INFO - Model summary saved at /srv/GadM/Datasets/Tmp/Hand-Gesture-Recognition/Model/stats/resnet34_frozen_model_summary.txt
epoch      train_loss    valid_loss   accuracy   time
0       0.507201     0.528244  0.832440  6:51:59
Better model found at epoch 0 with valid_loss value: 0.5282440185546875.
1       0.488131     0.508363  0.839103  7:10:15
Better model found at epoch 1 with valid_loss value: 0.5083632469177246.
2       0.458712     0.492844  0.843800  7:08:40
Better model found at epoch 2 with valid_loss value: 0.49284377694129944.
2024-11-15 09:29:33,319 - __main__ - INFO - Saving loss curve plot...
2024-11-15 09:29:33,904 - __main__ - INFO - Loss curve plot saved at /srv/GadM/Datasets/Tmp/Hand-Gesture-Recognition/Model/plots/resnet34_frozen_loss_curve.png
2024-11-15 09:29:33,904 - __main__ - INFO - Training complete.
2024-11-15 09:29:33,904 - __main__ - INFO - Starting evaluation...
2024-11-15 12:19:12,036 - __main__ - INFO - Classification report saved at /srv/GadM/Datasets/Tmp/Hand-Gesture-Recognition/Model/stats/resnet34_frozen_classification_report.txt
2024-11-15 12:19:13,912 - __main__ - INFO - Confusion matrix saved at /srv/GadM/Datasets/Tmp/Hand-Gesture-Recognition/Model/plots/resnet34_frozen_confusion_matrix.png
2024-11-15 12:19:13,912 - __main__ - INFO - Evaluation complete.
2024-11-15 12:19:13,912 - __main__ - INFO - Process completed successfully.
```

Abbildung 2: Trainings-Log für das ResNet34-Modell im Frozen-Modus

6 Evaluierung

Die Evaluierung des Projekts erfolgte anhand von Loss-Kurven, Klassifikationsberichten und Confusion-Matrizen. Ziel war es, die Unterschiede zwischen den beiden Trainingsansätzen (`frozen` und `unfrozen`) hinsichtlich ihrer Effizienz und Klassifikationsleistung zu analysieren.

6.1 Analyse der Loss-Kurven

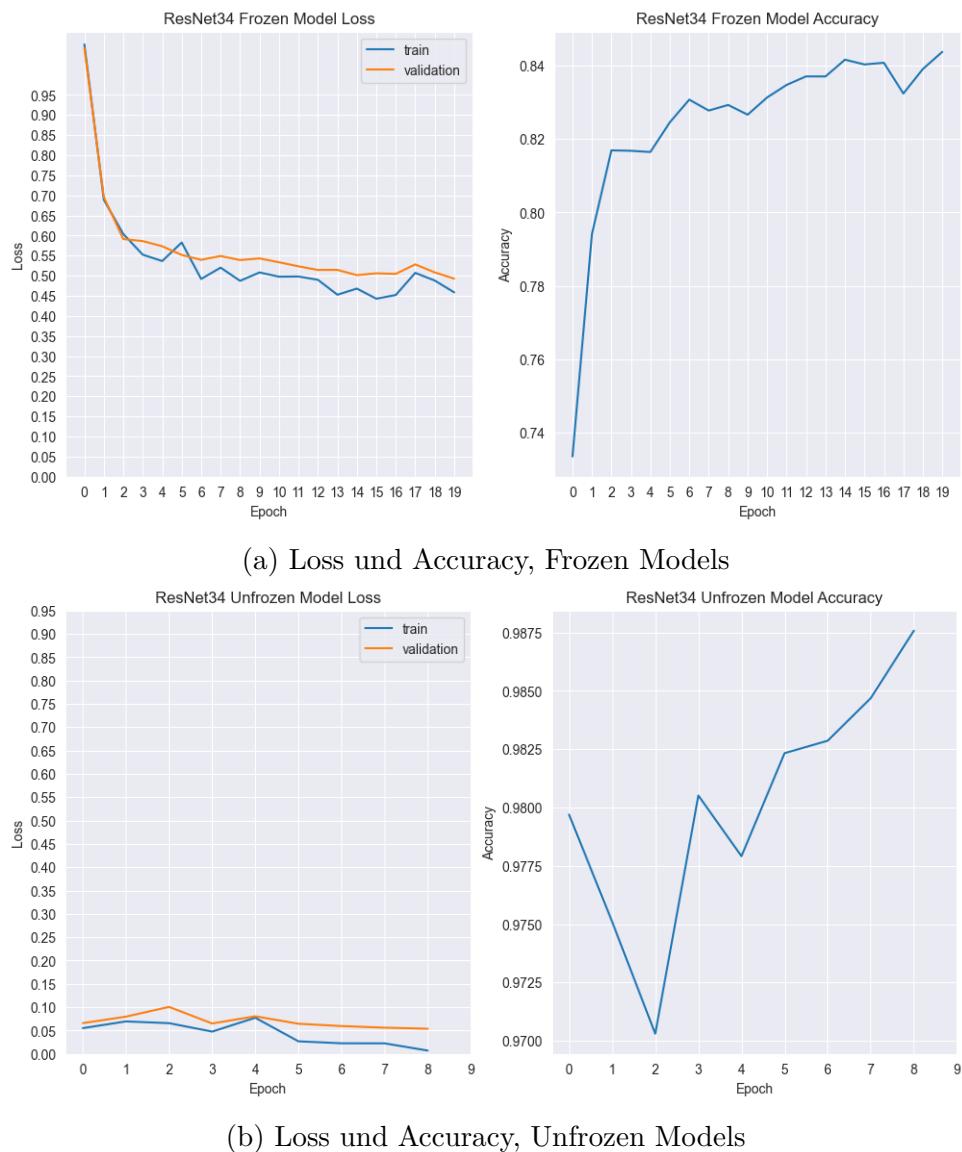


Abbildung 3: Vergleich der Loss- und Accuracy-Kurven für Frozen und Unfrozen Modelle.

Die Loss-Kurven (Abbildung 3) zeigen, dass das **frozen**-Setup zu Beginn einen hohen Verlust aufwies, der langsam abnahm und sich nach etwa 6 Epochen stabilisierte. Die finale Genauigkeit lag bei rund 84%. Im Gegensatz dazu startete das **unfrozen**-Setup mit einem niedrigeren Loss und konvergierte schneller. Es erreichte nach wenigen Epochen eine Genauigkeit von etwa 98%. Trotz längerer Trainingszeit pro Epoche (ca. 13 Stunden gegenüber 7 Stunden im **frozen**-Setup) war das **unfrozen**-Setup insgesamt effizienter.

6.2 Klassifikationsberichte und Performance-Metriken

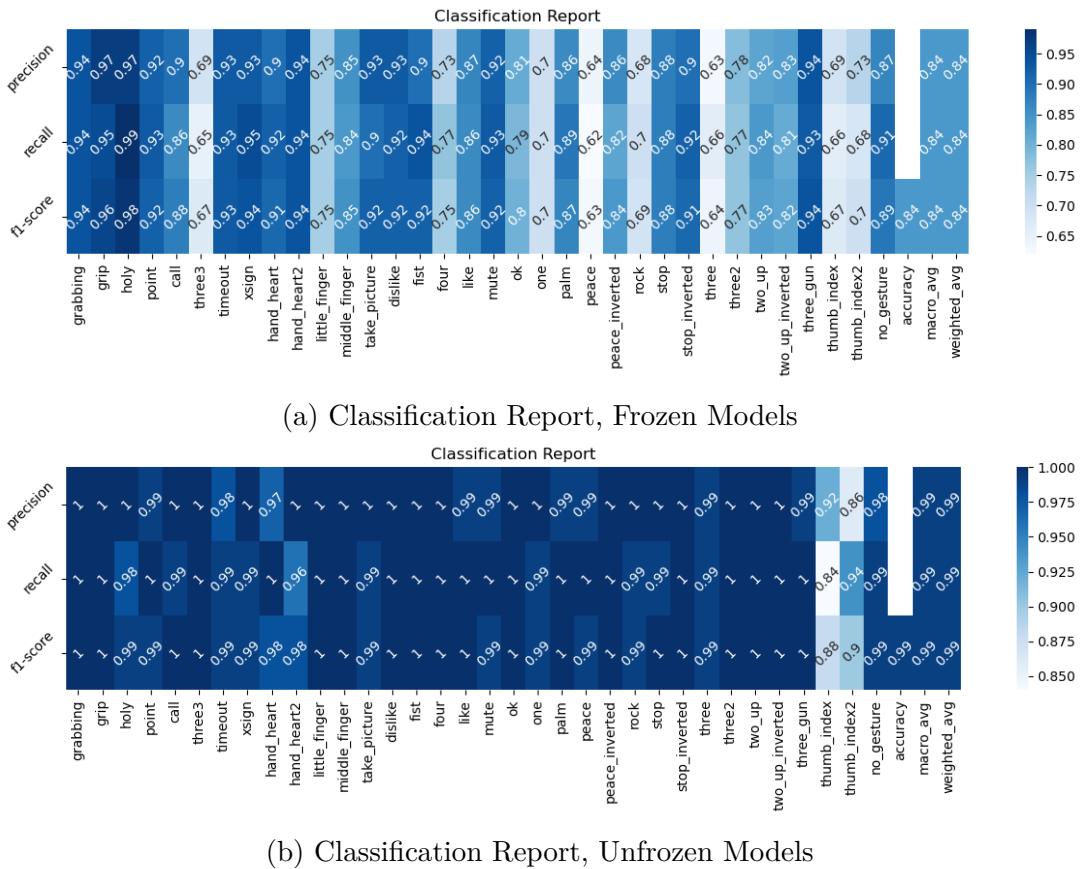
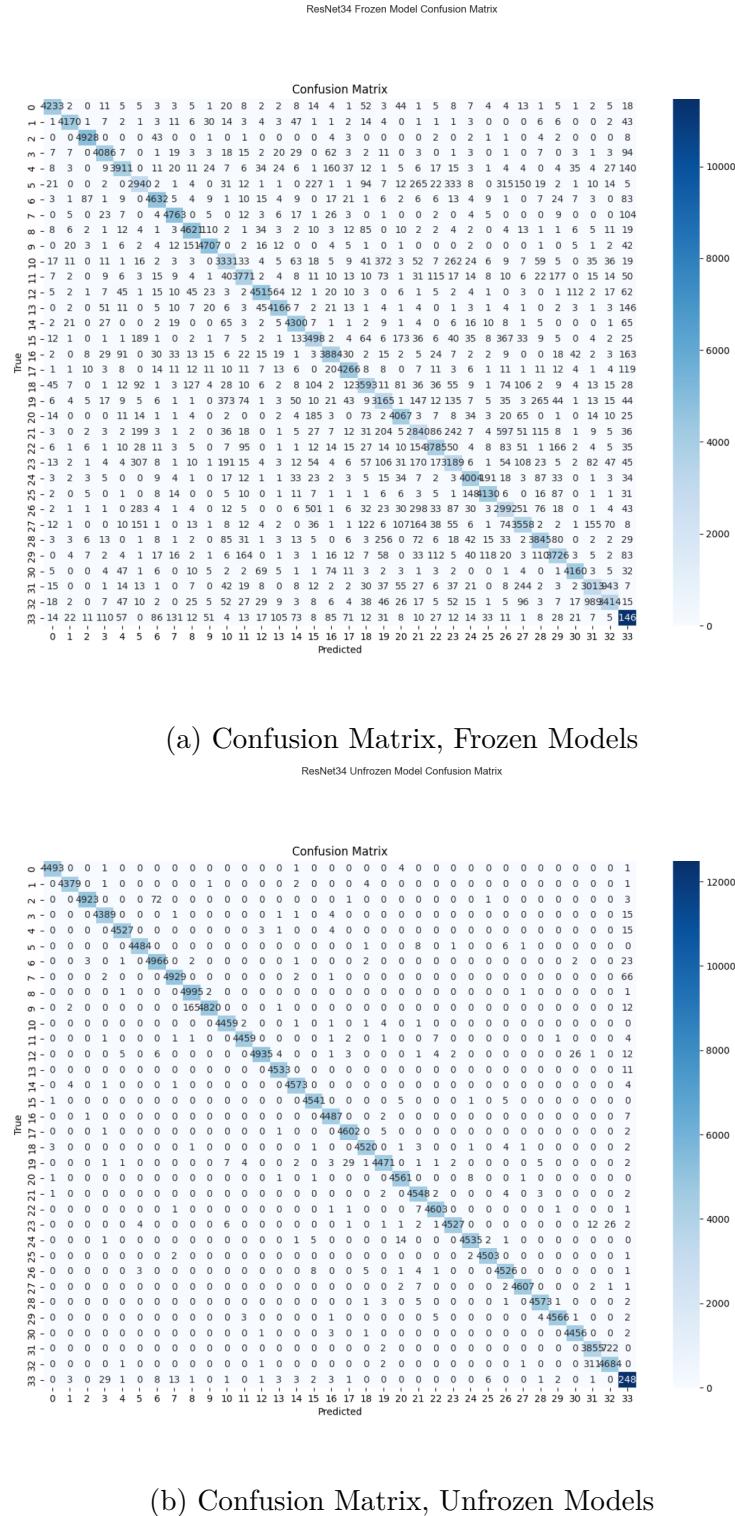


Abbildung 4: Vergleich der Klassifikationsberichte für Frozen und Unfrozen Modelle.

Die Klassifikationsberichte (Abbildung 4) verdeutlichen, dass das **frozen**-Setup eine durchschnittliche Genauigkeit von 84% erreichte, wobei einige Klassen wie *grabbing* und *holy* hohe Werte über 0,90 erzielten. Schwächere Ergebnisse waren jedoch bei Klassen wie *thumb_index2* sichtbar, die nur 0,60 bis 0,70 erreichten. Das **unfrozen**-Setup erzielte hingegen nahezu perfekte F1-Scores zwischen 0,99 und 1,00 für fast alle Klassen und eliminierte die Schwächen des **frozen**-Setups.

6.3 Confusion-Matrizen



(a) Confusion Matrix, Frozen Models

ResNet34 Frozen Model Confusion Matrix

(b) Confusion Matrix, Unfrozen Models

Abbildung 5: Vergleich der Confusion-Matrizen für Frozen und Unfrozen Modelle.

Die Confusion-Matrizen (Abbildung 5) zeigen, dass das `frozen`-Setup häufigere Fehlklassifikationen in den Off-Diagonalen aufweist, was auf eine geringere Fähigkeit hinweist, ähnliche Klassen zu unterscheiden. Das `unfrozen`-Setup zeigt eine fast perfekte Klassifikation mit minimalen Fehlklassifikationen und einer klaren Konzentration der Werte entlang der Hauptdiagonale.

6.4 Vergleich mit HAGRID-Baseline-Modellen

Die Performance des `unfrozen`-Setups ist bemerkenswert im Vergleich zu den HAGRID-Baseline-Modellen, die F1-Scores von etwa 98% mit ResNet18 und ResNet152 erreichten. Diese Baselines wurden jedoch mit einem Trainingsdatensatz von 100.000 Bildern pro Klasse trainiert, während unser `unfrozen`-Modell vergleichbare Ergebnisse mit nur 10.000 Bildern pro Klasse erzielte. Dies zeigt die Effizienz unseres Ansatzes bei der Nutzung begrenzter Datenmengen.

6.5 Zusammenfassung

Die Ergebnisse verdeutlichen, dass das `unfrozen`-Setup hinsichtlich Konvergenzgeschwindigkeit, Genauigkeit und Generalisierungsfähigkeit überlegen ist. Die zusätzliche Trainingszeit pro Epoche wird durch die signifikanten Leistungsgewinne gerechtfertigt. Der Vergleich mit den HAGRID-Baseline-Modellen zeigt, dass unser Ansatz eine vergleichbare Performance mit einem Bruchteil der Trainingsdaten erzielen konnte, was die Effektivität der verwendeten Strategie unterstreicht.

7 Demo

Für die praktische Umsetzung wurde ein Python-Skript erstellt, das einen Live-Kamerastream verarbeitet und in Echtzeit die Erkennung sowie Klassifizierung von Händen vornimmt. Die Ergebnisse werden direkt auf dem Videostream visualisiert, indem erkannte Hände mit einer *Bounding Box* umrandet und die zugehörige Klasse eingeblendet wird.

Durchzuführende Schritte:

1. **Handerkennung mit YOLO:** Für die Detektion der Hände wurde das bereits vortrainierte YOLO-Netzwerk aus dem Hagrid-Datensatz verwendet.
2. **Ausschneiden der Hände:** Mit den durch YOLO ermittelten *Bounding Boxes* wurden die Bereiche der Videoframes, in denen Hände erkannt wurden, ausgeschnitten. Dadurch konnten nur die relevanten Bildausschnitte weiterverarbeitet werden.
3. **Klassifizierung der Hände:** Die ausgeschnittenen Hand-Bilder wurden anschließend in das zuvor von uns trainierte Modell (unfrozen Setup) geladen, das für die Klassifizierung von Handgesten optimiert wurde.

In Abbildung 6 ist ein Beispiel einer Handdetektion zu sehen, bei der die Hände erfolgreich durch *Bounding Boxes* markiert und klassifiziert wurden. Auf einem ressourcenstarken System (Intel i7 13700, NVIDIA RTX 4080 und 64 GB DDR5 RAM) beträgt die Verarbeitungszeit eines einzelnen Frames etwa **250 ms**, was eine flüssige und performante Ausführung des Systems gewährleistet. Im Vergleich dazu benötigt ein durchschnittliches System (AMD Ryzen 7 8845HS, AMD Radeon 780M und 16 GB DDR5 RAM) rund **500 ms** pro Frame, was die Ausführung deutlich verlangsamt und keine wirklich flüssige Darstellung mehr ermöglicht.

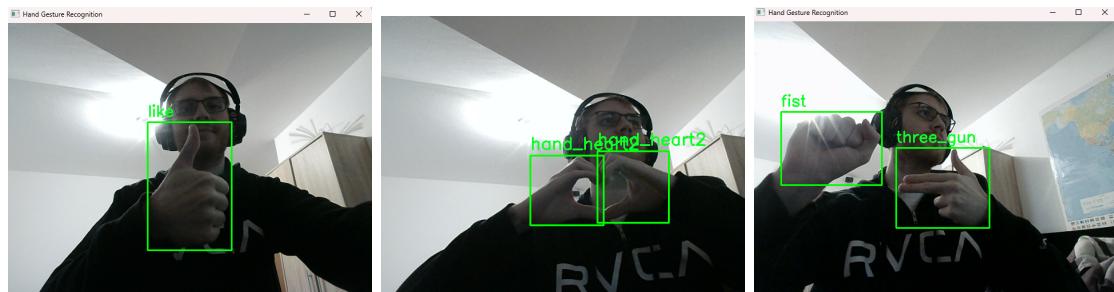


Abbildung 6: Demo Ausschnitte einer Handdetektion und Klassifizierung

8 Fazit

Das Projekt hat die zentrale Bedeutung eines fundierten Verständnisses der zugrunde liegenden Daten hervorgehoben. Ein klares Bild der Daten ist essenziell, um sinnvolle Ziele zu definieren und den gesamten Entwicklungsprozess darauf auszurichten. Die Datenvorbereitung stellte den ersten kritischen Schritt dar, da die Qualität und Struktur der Daten die Grundlage für den Erfolg eines jeden Deep-Learning-Modells bilden. Dieser Prozess, obwohl oft herausfordernd, erwies sich als unverzichtbar für die nachfolgenden Schritte.

Die Implementierung des Convolutional Neural Network (CNN) und das anschließende Training folgten einer bewährten Methodik, wobei die eigentliche Herausforderung darin lag, die Modelle systematisch zu evaluieren und die Ergebnisse präzise zu interpretieren. Hierbei wurde deutlich, dass die sorgfältige Analyse der Ergebnisse ebenso entscheidend ist wie die eigentliche Modellentwicklung. Nur durch eine umfassende Evaluierung konnten die Stärken und Schwächen der verschiedenen Ansätze aufgedeckt und Optimierungspotenziale identifiziert werden.

Das Projekt hat auch gezeigt, dass das Fine-Tuning eines gesamten Modells in bestimmten Anwendungsfällen zusätzliche Vorteile bieten kann. Insbesondere bei komplexeren Datensätzen oder Aufgaben, bei denen feine Details entscheidend sind, ermöglichen die Filter eines CNNs, tieferliegende Merkmale zu extrahieren und zu optimieren. Dies eröffnet Potenziale, die durch das Einfrieren der Schichten ungenutzt bleiben. In solchen Szenarien kann das vollständige Nachtrainieren des Modells nicht nur die Leistung verbessern, sondern auch die Flexibilität erhöhen, um auf spezifische Anforderungen der Anwendung einzugehen.

Insgesamt bot das Projekt eine umfassende Lernerfahrung, die die verschiedenen Aspekte eines modernen Deep-Learning-Workflows – von der Datenvorbereitung über die Modellimplementierung bis hin zur Evaluierung – anschaulich und praxisnah verdeutlicht hat. Die gewonnenen Erkenntnisse unterstreichen die Bedeutung einer fundierten Herangehensweise, die sowohl die Qualität der Daten als auch die Wahl der Modellkonfiguration in den Mittelpunkt stellt.

Literaturverzeichnis

- [1] A. Kapitanov u. a., »HaGRID – HAnd Gesture Recognition Image Dataset,« in *Proceedings of the IEEE/CVF Winter Conference on Applications of Computer Vision (WACV)*, Jan. 2024, S. 4572–4581.
- [2] Pytorch, <https://pytorch.org/>, Accessed: 2024-12-07.
- [3] J. Howard u. a., fastai, <https://github.com/fastai/fastai>, 2018.