Karlsruher Institut für Technologie

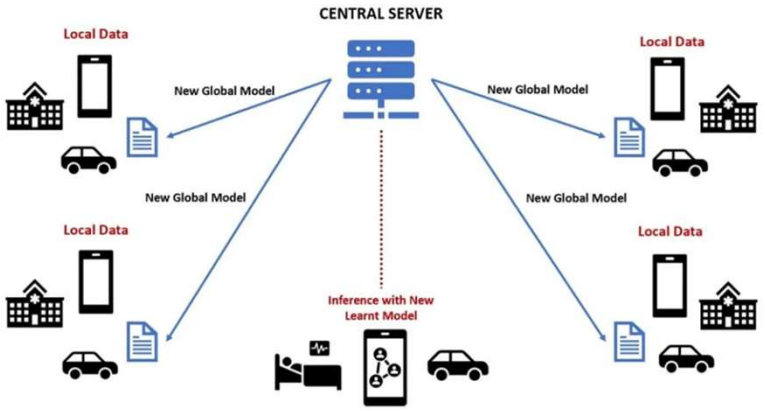
LAMA ItW: Federated Learning Classification

Einführung in & Experimente mit Federated Learning

Léo Brucker & Cyril Rudolph

Einführung: Federated Learning (FL)

- **Problemstellung:**
  - Normales Machine Learning: Alle Daten in einem Ort
  - Federated Learning: Daten über verschiedene Geräte (Clients) verteilt und können nicht zentral gespeichert werden



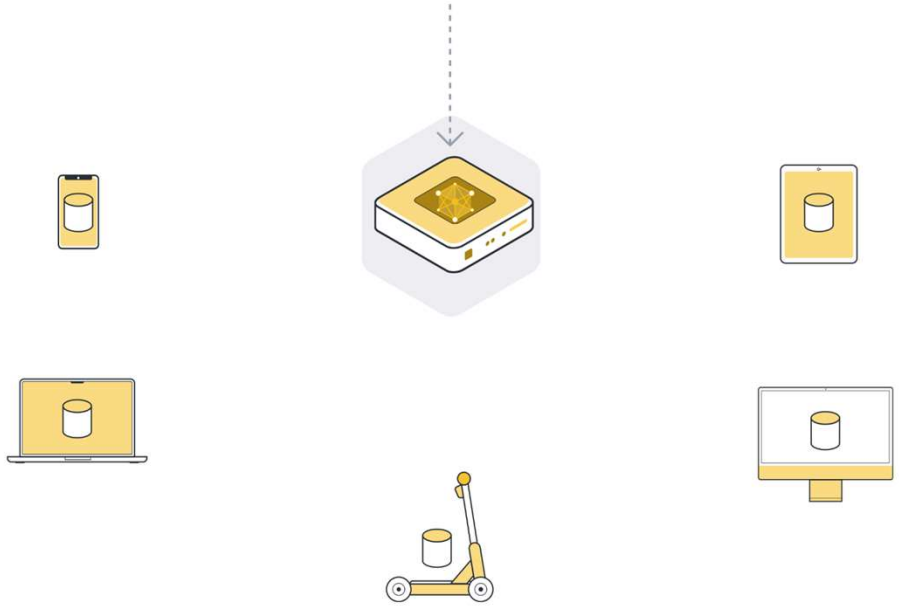
- **Motivation:**
  - Datenschutz (Bsp: Patientendaten im Krankenhaus oder persönlichen Schreibdaten auf der Handy-Tastatur)
  - Echtzeit-lernen: Modell wird kontinuierlich geupdatet und passt sich zu Änderungen und neuen Daten an.
  - Lokale Modelle: Jedes Modell auf Nutzer personalisiert

Einführung: Funktionsweise

- **Erinnerung:**
  - Convolutional Neural Network
  - Transfer Learning
- **Ablauf einer FL Epoche:**
  - Lokale Modelle trainieren
  - Aggregation der lokalen Trainingsergebnisse
  - Update des globalen Modells
- **Herausforderung**
  - Class Imbalance in lokalen und globalen Datensätzen

FL Step 1: Initialize Global Model

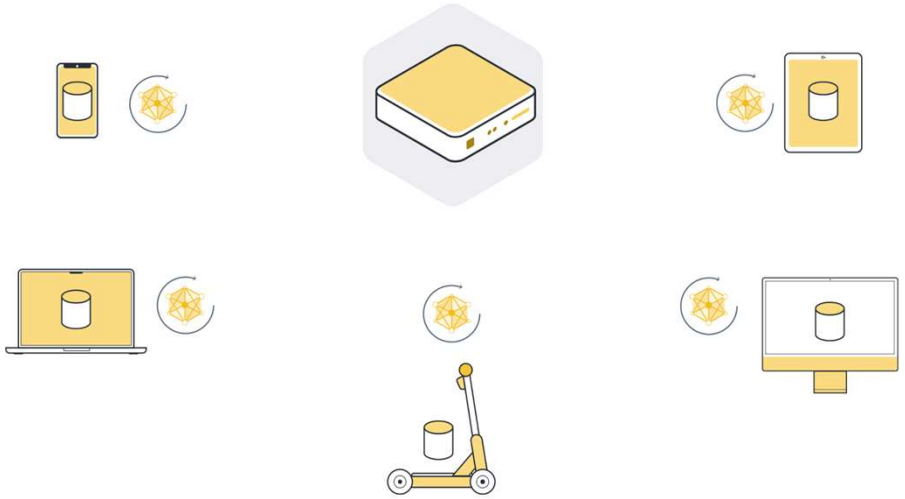
- Modell erstellen (CNN)
- Globales Modell initialisieren
- Parameter bestimmen
  - Anzahl an Clients berücksichtigen
  - Lernrate
  - Aggregations-Strategie



Source: https://flower.dev/docs/framework/tutorial-series-what-is-federated-learning.html

FL Step 2: Local Client Training

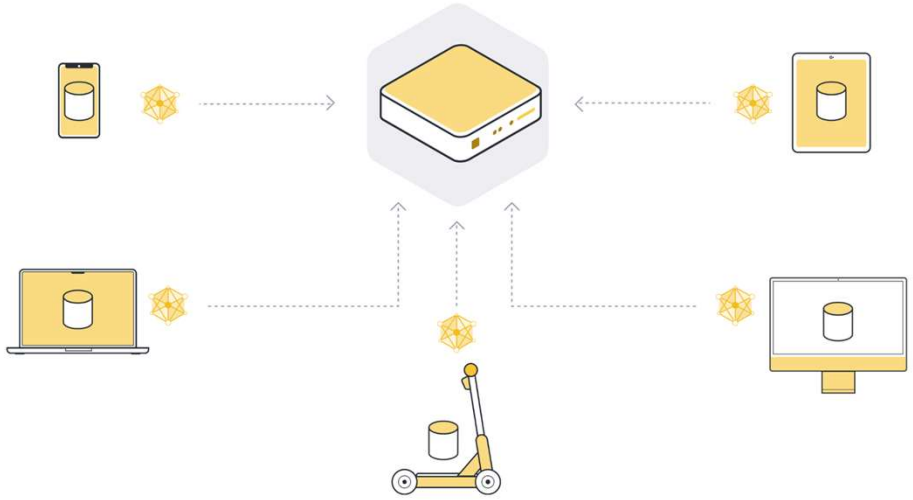
- Modell wird an Clients übergeben
- Bestimmte Anzahl ausgewählt, die das Modell trainieren
- Clients verändern die Parameter ihres lokalen Modells



Source: https://flower.dev/docs/framework/tutorial-series-what-is-federated-learning.html

FL Step 3: Aggregate Local Models

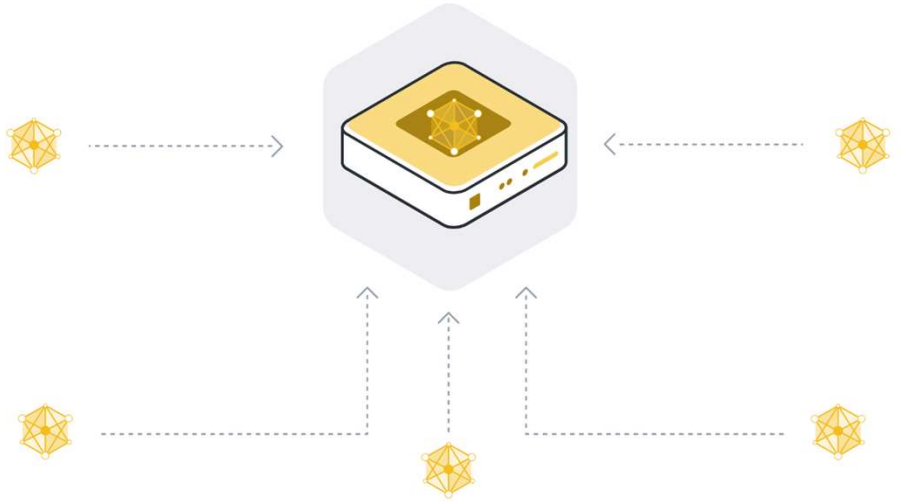
- Server erhält die Parameter von den lokalen Modellen
- Mit gewählter Strategie werden die Parameter zusammengeführt



Source: https://flower.dev/docs/framework/tutorial-series-what-is-federated-learning.html

FL Step 4: Update Global Model (& Repeat)

- Update des globalen Modells durch die erfolgreiche Aggregation
- Kritische Punkte:
  - Schlechtere Leistung im Vergleich zu klassischem Transfer Learning
  - Datensicherheit gewährleisten und trotzdem Leistung verbessern



Source: https://flower.dev/docs/framework/tutorial-series-what-is-federated-learning.html

Einführung: Unsere Aufgabe

- Federated Learning Model für EMNIST-Datensatz erstellen
- Reduktion der lokalen Komplikationen
  - Class Imbalance messen können (MID, WCS)
  - Parameter des Models verändern (z.B. Batchsize, Clientnumber …)
  - Versuchen das FL Modell gegen das Centralized zu konvergieren
- Erkenntnisse auswerten

Vielversprechende Parameter

FL-Modell:

- Batchsize
- CNN Model
- Learning Rate
- Epochs
- Fraction Train
- Aggregate Strategy

Simulation:

- Anzahl Clients
- MID & WCS

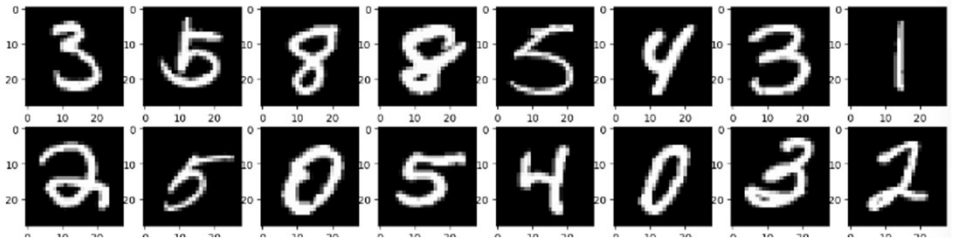
Umsetzung: Vorlage

Karlsruher Institut für Technologie

- Viele Iterationen mit unterschiedlichen Parametern notwendig
- Eine Vorlage, in der wir die Parameter schnell anpassen können
- Speichert automatisch die Ergebnisse in Dateien
- Auswertung und Interpretation für die nächsten Modelle bleibt manuell ☺

```
##   MAIN  PARAMS   ##
######################
CENTRALIZED = False  ## False if Federated
SPLIT = "digits" # Choose from "digits, balanced"...
MODEL = "internet" # Choose from "chatgpt", "internet", "tutorial"

######################

## OTHER GLOBAL PARAMS ##
PROGRESS_BAR = CENTRALIZED # set to "CENTRALIZED" to only progress bar
USE_GPU = True # False for CPU
VALIDATION_SPLIT = 0.1
OPTIMIZER = "sgd" # Choose from "sgd" and "adam"
CRITERION = nn.CrossEntropyLoss()

### CENTRALIZED PARAMS ###
LR_CENTRALIZED = 0.01  # Learning rate
MOM_CENTRALIZED = 0.9 # Momentum
EPOCHS_CENTRALIZED = 3 # Epochs
BATCH_SIZE_CENTRALIZED = 256

### FEDERATED PARAMS ###
LR_FEDERATED = 0.01  # Learning rate
MOM_FEDERATED = 0.9 # Momentum
EPOCHS_FEDERATED = 1 # Epochs
BATCH_SIZE_FEDERATED = 128
```

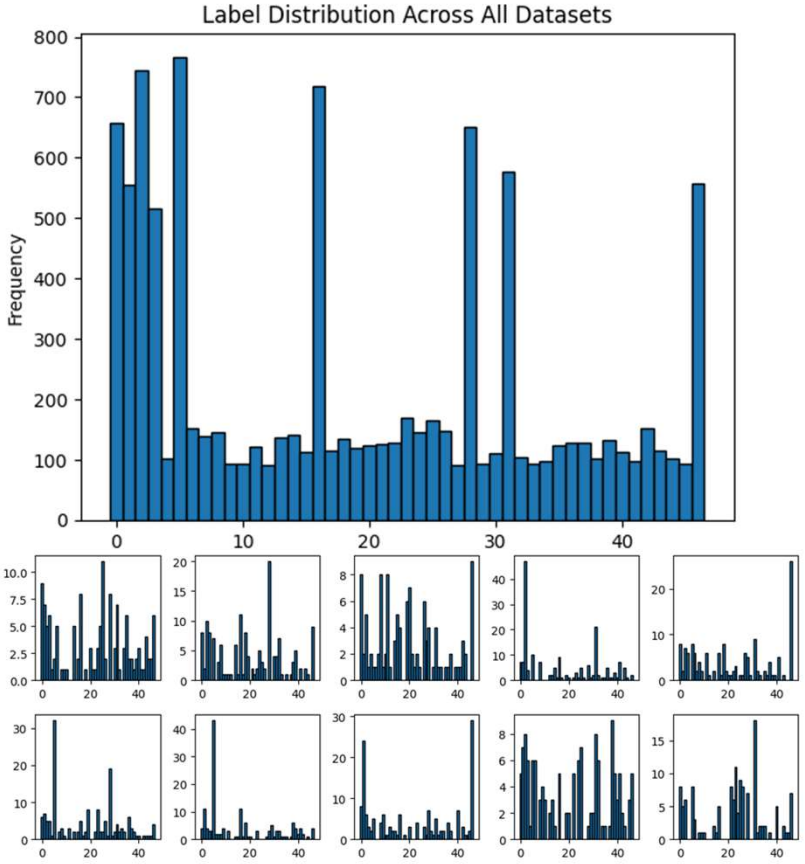
Umsetzung: EMNIST & FEMNIST



```python
# Create Federated Dataset
def get_federated_loaders():
    trainset, testset = get_dataset()

    num_images = len(trainset) // NUM_CLIENTS
    partition_len = [num_images] * NUM_CLIENTS

    trainsets = random_split(trainset, partition_len)
    #
    global TRAINSETS
    TRAINSETS = trainsets
```

- Datensatz: EMNIST
- keine Datenaufbereitung notwendig
- Gute Baseline erstellbar
- Aufteilung des Datensatzes für FEMNIST (siehe links)
- Class Imbalance Simulation möglich
- Verschiedene Splits („mnist“, „digits“)

Umsetzung: Class Imbalance Creation



- Class Imbalance erstellen wir selber
- Parameter für globale und lokale Imbalance
- (links) Globale und lokale imbalance Parameter sehr hoch gestellt

MID & WCS

Source: https://arxiv.org/abs/2109.04094v2

Multiclass Imbalance Degree

- Messung des globalen Klassen Ungleichgewicht
- Verhältnis zwischen dominierender & minoritärer Klasse unzureichend
- Maßeinheit (zwischen 0 & 1), die insbesondere die Größe der einzelnen Klassen eines Datensatzes betrachtet

Weighted Cosine Similarity

- Messung des Zusammenhangs zwischen globalen & lokalen Ungleichgewicht (von Klassen)
- Erweiterung von MCS, hatte nicht die Datensatzgröße berücksichtigt
- Gibt Werte zwischen 1 - 1/sqrt C

Hat am Ende des Tages uns nicht viel weitergeholfen..
die Werte bleiben selbst bei sehr ungleichmäßigen Datensätzen
sehr nah an 0 bzw. 1 und sagen nicht soviel aus wie wir es uns gerne erhofft hätten

Ergebnisse

Scenario 1: MNIST

→ Parameter Optimierung

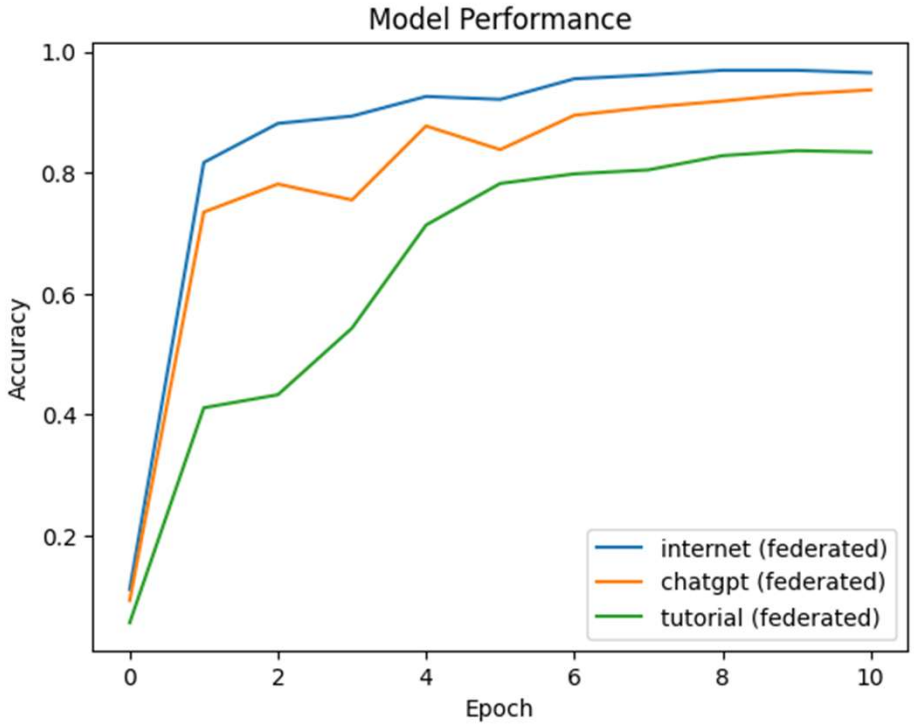
- Model: ?
- Batch Size: ?
- Learning Rate: ?
- Strategy: ?

Ergebnisse

Scenario 1: MNIST

→ Parameter Optimierung

- Model: Internet
- Batch Size: ?
- Learning Rate: ?
- Strategy: ?

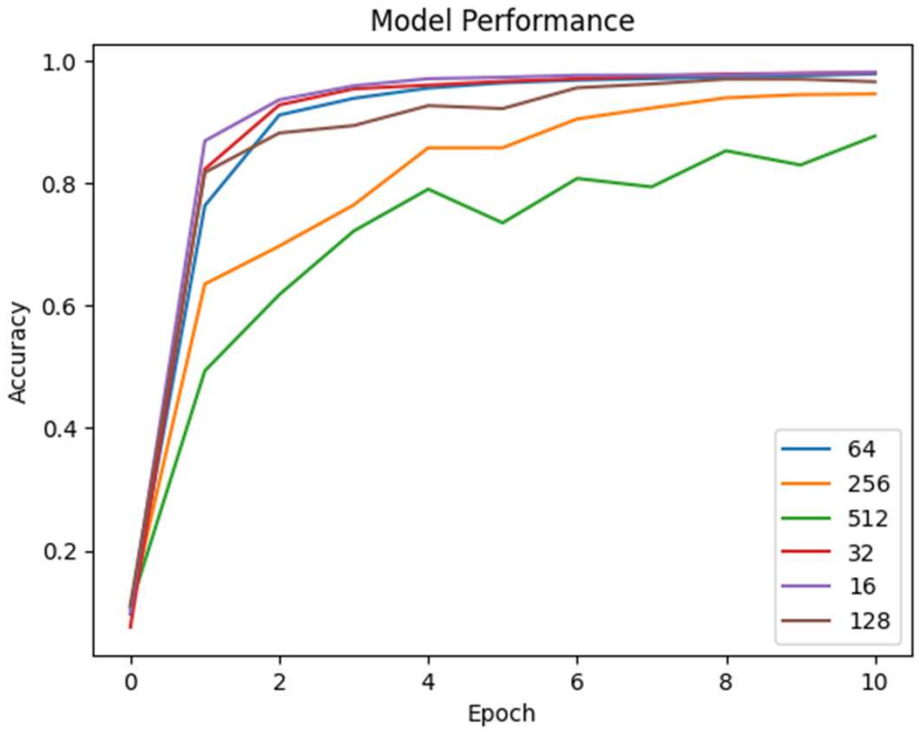


Ergebnisse

Scenario 1: MNIST

→ Parameter Optimierung

- Model: Internet
- Batch Size: 32
- Learning Rate: ?
- Strategy: ?

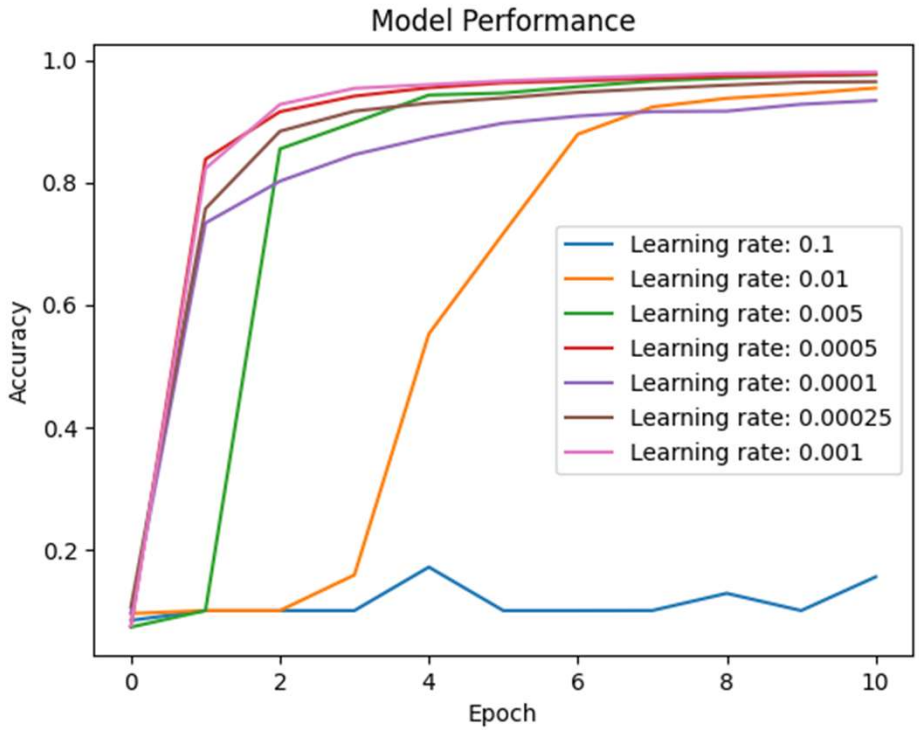


Ergebnisse

Scenario 1: MNIST

→ Parameter Optimierung

- Model: Internet
- Batch Size: 32
- Learning Rate: 0.001
- Strategy: ?



Ergebnisse

Scenario 1: MNIST

→ Parameter Optimierung

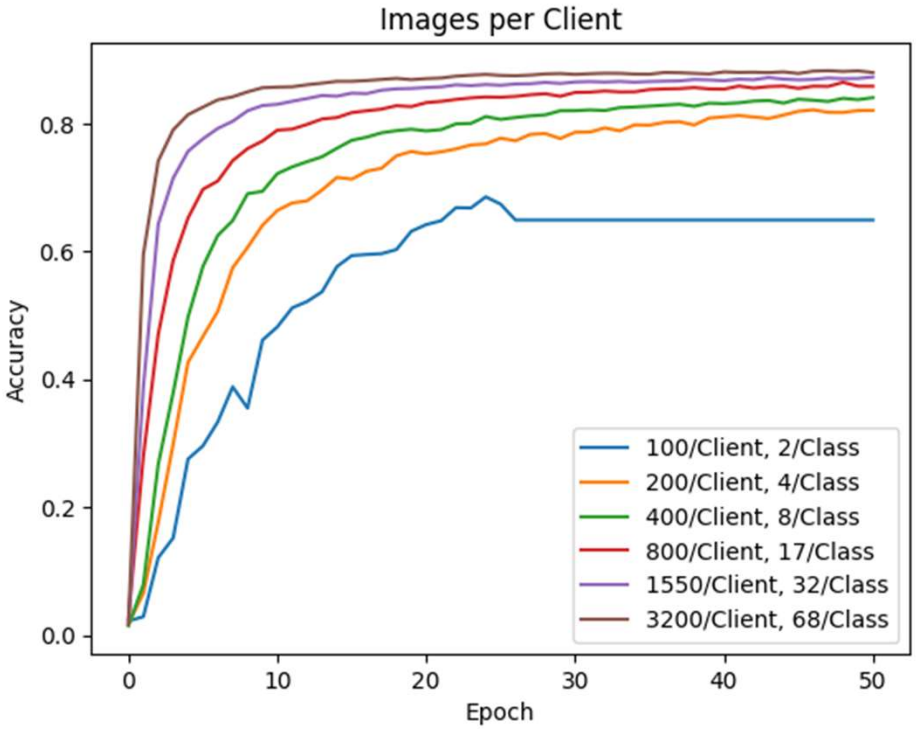
- Model: Internet
- Batch Size: 32
- Learning Rate: 0.001
- Strategy: FedAvg

FedAvg: 98, 1%
FedAdam: 4,0% uff
Andere: ~97%
→also wenig Einfluss

Ergebnisse

Einfluss von Client Größe:

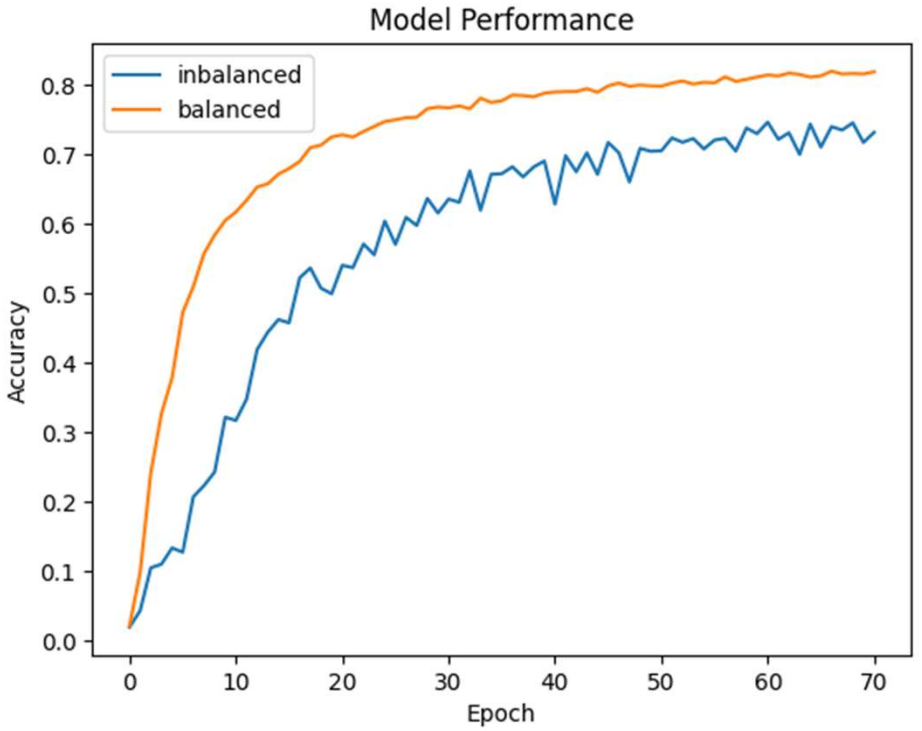
- Desto mehr Images/Client, desto näher an Centralized Baseline
- Clients, Fraction_Train nicht einfach generell vergleichbar, Datensatz abhängig



Ergebnisse

Einfluss von Imbalance

Imbalance beeinflusst wie erwartet das Ergebnis negativ



- **WCS:**
  - Balanced: 1.0
  - Unbalanced: 0.74
- **MID:**
  - Balanced: 0.0
  - Unbalanced: 0.09

Ergebnisse

Global oder Local Imbalance?

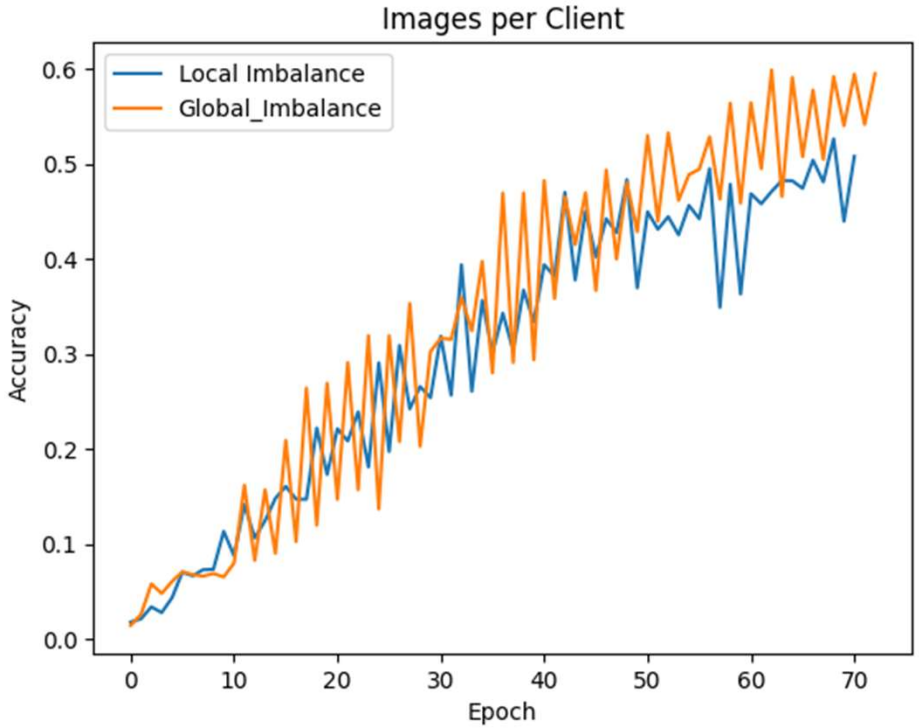
Local Imbalance Scenario:

WCS: **0,66** MID: **0,07**

Global Imbalance Scenario:

WCS: **0,9** MID: **0,14**

→ ob die imbalance global oder local ist macht kein großen unterschied.



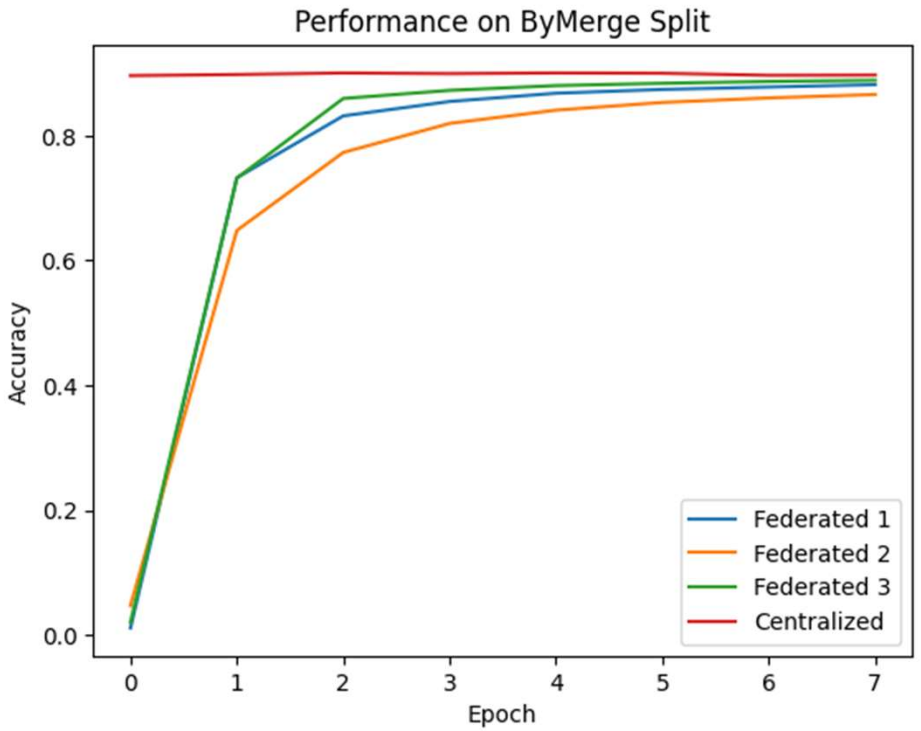
On a “real” Scenario

Testen unserer Modells auf den “ByMerge” Split, der schon vorunbalanced ist.

Conclusion:

Unserer FL funktionieren fast genau so gut wie die centralized Variante!

Und respektiert Privatspähre ☺



Herausforderungen während des Projektes

- mit GIT rumgeschlagen
- Lange versucht mit einer momentan kaum benutzbaren Bibliothek zu arbeiten (Warnung vor Tensorflow_Fed)
- Fehlende Orientierung im Projekt

Was haben wir daraus gelernt?

- Klarere Zielsetzung
- Nutzen-Kosten-Analyse (auf Zeit bezogen)

Zusammenfassung:

Zentrale Ergebnisse:

- Parameter mit Einfluss
  - CNN-Modell
  - Learning Rate
  - Batch Size
- Schwierig zu verallgemeinern
  - Anzahl an Clients
  - Anzahl der genutzten Clients
- Auf Unbalanced Dataset funktioniert unser angepasstes Modell gut

Mögliche Fortführung des Projektes:

- Kleingruppen Agglomeration
- Eigene Aggregate Strategie erstellen, Strategien noch mehr anpassen
- Erstellte Ergebnisse genauer auswerten, deutlich mehr Parameterkombinationen ausführen → größerer Datensatz