Towards Real World Federated Learning

Project Summary

TAs: Debora Caldarola, Eros Fanì

Many data nowadays cannot be used by traditional Machine Learning (ML) approaches because of their sensitive and privacy-protected nature, even if their use could significantly improve our models' performance. A concrete example is provided by medical data: doctors may use the knowledge from a trained AI model, yet patient data cannot be collected because it is inherently private. Introduced by Google, **Federated Learning** is an ML scenario aiming to learn from privacy-protected data without violating the regulations in force. Within a client-server architecture, a server-side global model has to be learned by leveraging the clients' data, without breaking their privacy. This is achieved by never giving the server direct access to the data and transferring the trained model parameters.

Let us now imagine the following use case. Alice likes traveling and taking many pictures of the places she visits with her smartphone. On the other hand, John is a professional photographer, so he rarely uses his phone and mainly relies on his camera. Martha is an influencer showing Rome to her followers, so her phone is packed with pictures of that city. All of them would like to have access to an AI model able to recognize the place in which their photos were taken. Due to the private nature of their pictures, they need a federated scenario. Locally, the model will have access to Alice's pictures of many places, almost no photos on John's phone, and Martha's pics from Rome. ***What happens when a model is trained on so differently distributed data?***

In addition, the pictures could be taken in the daytime or at night, during sunny or rainy days *i.e.* with different weather and light conditions, and so on. All these variables make learning harder. That implies additional issues arise when moving towards more realistic scenarios, which have to be explicitly addressed. In this case, ***can the model generalize to different and potentially new visual shifts, i.e. domains?***

Lastly, it is not realistic to assume to have access to labeled data on the client-side, since labeling is costly and often requires domain expertise. ***What happens when the client's data is not labeled?***

Projects overview

The goal of this project is to become familiar with the Federated Learning (FL) framework and the issues arising when facing more realistic scenarios.

Track 2

TA: Eros Fanì

- Task: **semantic segmentation** for autonomous driving.
- Challenges: heterogeneously distributed data and extension to unlabeled data on the client side.

Main goals:

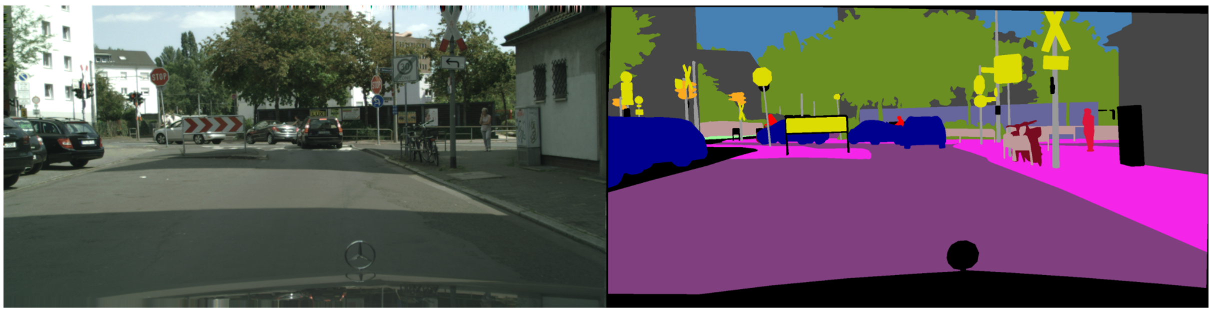
- To become familiar with the standard federated scenario and its state of the art
- To understand the real-world challenges related to statistical heterogeneity
- To experiment on a more realistic scenario which does not require client-side labeled data
- To replicate the experiments detailed in the following sections
- To implement and test your contribution for solving one of the highlighted issues

You will be provided with both existing code and federated datasets to start running the first experiments. *Details follow.*

TRACK 2: Federated Learning for autonomous driving

PROJECT OVERVIEW AND GOALS

Semantic Segmentation (SS) is essential to make self-driving vehicles autonomous, enabling them to understand their surroundings by assigning individual pixels to known categories. However, it operates on sensible data collected from the users' cars; thus, protecting the clients' privacy becomes a primary concern.



To this end, Federated Learning (FL) constitutes a possible solution. FL is a relatively new machine learning paradigm aiming to learn a global model while preserving privacy and leveraging data on millions of remote devices.
However, most of the existing works on FL unrealistically assume labeled data in the remote clients. Thus, the new task of **Federated source-Free Domain Adaptation (FFreeDA)** has recently been introduced. In FFreeDA, the clients' data is unlabeled, and the server accesses a source-labeled dataset for pre-training only.

In this project, you will dive into the motivations and applications of **Federated Learning (FL)** to the **Semantic Segmentation (SS)** task applied to **self-driving cars**. The main objective of this project is to understand what are the FL framework and the SS task and become familiar with them. You will learn the motivations behind applying FL to SS in the context of autonomous vehicles and how to simulate experiments to resemble real-world scenarios using the **PyTorch library**. Moreover, you will understand the difficulties and challenges of this scenario and propose your solutions. Finally, you will be introduced to the new task of **Federated source-Free Domain Adaptation** applied to the Real-time Semantic Segmentation networks (**FFreeDA**).

BEFORE STARTING

Suggestion: read this document at least once before starting with the experiments. Be sure to have at least an idea of what you are going to implement.

Take your time to familiarize yourself with SS and FL, by reading the following mandatory papers. Before preceding, be sure to have understood these concepts. You can find the references at the end of this document.

• Semantic Segmentation: [1], [2], [3], [4], [5]
• Federated Learning: [6], [7]

Feel free to explore more online resources later (you are strongly encouraged to enrich your knowledge by exploring more related papers by yourself while waiting for your experiments to be completed!).
Moreover, you should be able to perform experiments using the PyTorch library. You might refer to this guide to get started if you need it.

Codebase

The starting codebase for your experiments can be found at this link.
The code is organized as follows:

- in `data/` you will download the dataset. Instructions can be found in the following section
- in `utils/` you can find some useful functions
- the `main.py` file orchestrates the federated training
- in `client.py` and `server.py` you can find the classes containing useful functions for local training and server-side orchestration and models aggregation respectively.

Please, be sure to be familiar with the code and have a good understanding of its main parts *before* starting your experiments.

Resources and datasets

You can run your experiments on CoLab for free. Here you can find some suggestions to set up your working environment for this project:

- Download the datasets you will use from here and here and extract them. In the first directory, you will find a subset of the IDDAv3 dataset [3]. In the second directory, you will find a subset of the Cityscapes dataset and a subset of the GTA5 dataset [4]. In this project, you are required to use only the IDDA and GTA5 datasets (you can ignore the subset of Cityscapes). In the following of this document, we will refer to these subsets of the original datasets simply as IDDA and GTA5
- **Convention**: we refer to a ***domain*** for the IDDA dataset as a triad (town, weather/illumination, car), where in our case town ∈ {T01, T02, T03}, weather/illumination ∈ {Clear Noon, Clear Sunset, Hard Rain Noon} and car ∈ {Audi TT, Ford Mustang, Jeep Wrangler Rubicon} (Refer to the official IDDA website for more insights on this).
- Upload the dataset folders into your Google Drive account if you plan to perform your experiments on CoLab
- First, mount your Google Drive folder by executing the following code to access the dataset folder on CoLab:
```
from google.colab import drive
drive.mount('/content/drive')
```

- When executing on CoLab, remember to activate the GPU hardware accelerator: Runtime → Change runtime type → GPU
- If you are not planning to run your experiments on CoLab but on you plan to run on your own resources, you can install all needed packages using Anaconda or Miniconda by running
  ```
  conda env create -f mldl23fl.yml
  ```

Refer to the following additional resources to take inspiration for the coding implementations and other details:

- FedDrive official website
- FedDrive official code
- LADD official code

N.B. For easier comprehension, follow the project steps first and explore these resources only after you have read and understood the associated paper.

N.B. The above codes exploit the DistributedDataParellel (DDP) package to work on multi-GPUs and the WandB platform for logging the results. If you are working on CoLab, you can drop the DDP package in your implementation. You may log the statistics of your experiments using WandB if you feel comfortable with it (see below). However, this is not mandatory: any visualization tool is allowed (even text only. But plotting your curves is better for both you and us, to understand better what is happening in your experiments!).

N.B. Run your experiments for a reasonable number of epochs/rounds but consider that we are aware of the limited amount of resources at your disposal. Reduce the number of epochs/rounds if your experiments take too long, but be consistent with your choices in all the steps of the project. Ideally, you should run your experiments until convergence.

Useful additional references

All the experiments will be run using **PyTorch**. The official documentation together with useful tutorials can be found here.

In addition, if interested here you can find the official site of **Weights and Biases** for keeping track of your experiments and plotting the results. You first need to set up your team account and then install the `wandb` package and import it in the project. Detailed setup instructions can be found here.

1st STEP) CENTRALIZED BASELINE

In this step, you must implement a centralized baseline for the IDDA dataset using the DeeplabV3 network with MobilenetV2 as the backbone (the modules are already provided with the starting code).

Try some combinations of hyper-parameters and data augmentation for the same amount of epochs, and choose the best combination according to the mIoU metric.

You must use the train/test partitions provided within the IDDA folder:

- Use the `train.txt` file to retrieve the train images from their paths
- In the `test_same_dom.txt` file there is a list of other paths of images belonging to the same domains of the training images
- In the `test_diff_dom.txt` file there is a list of other paths of images belonging to different domains

N.B. In SS, the transforms applied to the image should reflect on the corresponding label. In other words, you need to be sure to avoid misalignment between the pixels of the image and the pixels of the corresponding labels. We suggest using the custom Torchvision transforms we provided with the starting code, or the custom OpenCV transforms you can find here.

Provide at least the following information in your final report:

Hyper-parameters	Set of transforms (data augmentation)	Number of training epochs	Eval mIoU (train partition)	Test Same Dom mIoU	Test Diff Dom mIoU
…	…	…	…	…	…

- How did you choose the hyper-parameters? Why?
- Discuss your results

2nd STEP) SUPERVISED FEDERATED EXPERIMENTS

It is time to build your first FL + SS experiments. For the following experiments, use the hyper-parameters and the data augmentation you have found in the previous step (and the same model and dataset).

Read [8] and familiarize yourself with the FL+SS task. If you want and need it, now it is the time to familiarize yourself with the code of FedDrive provided in the *Resources and datasets* section. Run FL+SS supervised experiments in the following configurations, using the FedAvg algorithm. You must use the `train.json` file provided with the IDDA dataset.

Provide at least the following information in your final report:

Clients per round	Number of local epochs	Number of rounds	Eval mIoU (train partition)	Test Same Dom mIoU	Test Diff Dom mIoU
2	1	…	…	…	…
2	3	…	…	…	…
2	6	…	…	…	…
4	1	…	…	…	…
8	1	…	…	…	…
…	…	…	…	…	…

- Discuss your results

3rd STEP) MOVING TOWARDS FFreeDA - PRE-TRAINING PHASE

In this step, you will finally move towards a more realistic scenario. In the real world, self-driving cars do not have access to ground-truth labels. If they had ground-truth labels, a SS model would not be necessary. Moreover, manually labeling the images is a costly process, and it is not realistic to assume that the clients have access to the labels associated with the images they collect. However, it is reasonable to assume that the model is pre-trained on an open-source supervised dataset as GTA5.

1) Read [9], [10], and [11] to take a cue on how to deal with the Domain Adaptation task in FL for SS, what are the main challenges, and the real-world motivations behind it.
2) Use the images listed in the train.txt file of the GTA5 dataset to train the network from scratch. **Pay attention to the semantic classes**. You have to select just the 16 in common with IDDA. Use the following mapping for the labels of GTA5 (to match the ones of IDDA):

```
class_map = {
   1: 13,  # ego_vehicle : vehicle
   7: 0,   # road
   8: 1,   # sidewalk
   11: 2,  # building
   12: 3,  # wall
   13: 4,  # fence
   17: 5,  # pole
   18: 5,  # poleGroup: pole
   19: 6,  # traffic light
   20: 7,  # traffic sign
   21: 8,  # vegetation
   22: 9,  # terrain
   23: 10,  # sky
```

```
    24: 11,   # person
    25: 12,   # rider
    26: 13,   # car : vehicle
    27: 13,   # truck : vehicle
    28: 13,   # bus : vehicle
    32: 14,   # motorcycle
    33: 15,   # bicycle
}
```

While training on GTA5, evaluate your model on the train set of IDDA, and check the performance on the two test sets. Tune your hyper-parameters and data augmentation again.

- Evaluate the model several times while training for every experiment (e.g., every t rounds). Save the checkpoint of your best model among all the evaluations of all the experiments you performed (i.e. not necessarily the model you obtained at the end of the last epoch). But don't be greedy! There is a lot of randomicity in the training procedure (especially if you do not fix the random seed), thus oscillations in the results are typical. Choose the best model carefully by observing the trend in your performance curves, and justify your choices.
- Check the performance on the two test sets of IDDA for your best model.

3) Read [12] to familiarize yourself with the FDA technique.
4) Apply FDA as in [11]: extract the average style from each client of the IDDA train set and form a bank of target styles. Then, using the hyper-parameters you found on the previous point, train the network from scratch again, as in point 2 of this list. However, now substitute every source image style (GTA5 dataset) with a random one from the bank of target styles during the training (you can think of this operation as a new transform). Evaluate again your results on the train set of IDDA, and check the results on the two test sets. Test some different FDA window sizes.

Provide at least the following information in your final report:

Hyper-parameters (in particular, do not forget the FDA window size!)	**Set of transforms (data augmentation)**	**Number of epochs**	**Eval mIoU (Source Dataset, train partition)**	**Eval mIoU Target Dataset (train partition)**	**Test Same Dom mIoU Target Dataset**	**Test Diff Dom mIoU Target Dataset**
…	…	…	…	…	…	…

- How did you choose the hyper-parameters? Why?
- Discuss your results

4th STEP) FEDERATED SELF-TRAINING USING PSEUDO-LABELS

Pseudo-labels can be used as an unsupervised (or semi-supervised) self-training technique to improve performance. In short, to get the pseudo-labels from a batch of images, make a forward pass through a *teacher model*, normalize the predictions and generate a label image for each input image. This label will be used as a ground-truth label by a *student model* during the training (it can also be done in real-time, i.e. without physically storing the pseudo-labels). For each pixel of an image, the teacher model will predict an array of confidence probabilities. If the highest probability is above a certain threshold, the final label is the one with the highest probability. Otherwise, if none of the predictions is above the threshold, the pixel is labeled as *don't care*. Hint: take inspiration from this part of the LADD code for this step.

1) Read [13] to familiarize yourself with pseudo-labels
2) Load the best checkpoint from step 3.2 and perform a FedAvg training on IDDA using pseudo-labels as ground-truth labels for your model, **without** using the original labels for training:
    - Before the training starts, set student model = pre-trained model, teacher model = pre-trained model
    - Perform FedAvg on the student model using the pseudo-labels generated by the teacher model as ground truth labels. Use the cross-entropy loss. Test the following three strategies to update the teacher model:
        i) teacher model never updated
        ii) teacher model = server model at the beginning of each round
        iii) teacher model = server model every T>1 rounds (the value of T is your choice. You can also test more than one possible T) (this point is not mandatory, but recommended)
3) Repeat point 2, but now use the best checkpoints from 3.4

Provide at least the following information in your final report:

Clients per round	Number of local epochs	Teacher model update interval	Number of rounds	Eval mIoU Target Dataset (train partition)	Test Same Dom mIoU Target Dataset	Test Diff Dom mIoU Target Dataset
2	1	∞	...	...	...	...
8	1	∞	...	...	...	...
2	1	1	...	...	...	...
8	1	1	...	...	...	...
2	1	t1	...	...	...	...
8	1	t1	...	...	...	...

2	1	t2	...	...	...	...
8	1	t2	...	...	...	...
...	...	...	...	...	...	...

- Discuss your results

5th STEP) IMPROVEMENTS (optional, but recommended)

Take inspiration from [8] and [11] to improve your work! We propose a list of possible ideas, but you are strongly encouraged to test yours! **Before starting, contact me to explain your ideas and intentions first, even if your idea is to pick one of the following listed below.** When not explicitly specified, you can find the references of the following bullet points within [8] and [11].

Legend:
- * represents the (minimum) difficulty level of the corresponding task
- * + * represents the expected maximum difficulty level of the corresponding task

Possible ideas:
- [***] Implement Domain Adaptation techniques based on the Batch Normalization layers to address the Statistical Heterogeneity problem (see SiloBN, FedBN)
- [*] Propose other train/test partitions and federated splits. Justify your choices
- [**] Repeat steps 3 and 4 using a subset of the Cityscapes dataset as the target dataset (already provided with GTA5)
- [****] Try a clustering scheme as in LADD
- [****] Implement regularization techniques as in LADD
- [*****] Test FL algorithms other than FedAvg (e.g. FedProx [14], SCAFFOLD [15], your personalized aggregation scheme...)
- [***] Test with some server optimizers [16]. First, verify the equivalence of SGD with momentum = 0 and learning rate = 1 with FedAvg.
- [*] Repeat some steps with another lightweight network, e.g. BiseNet V2 (look here for a possible implementation)
- [*****] Explore Satellite image datasets online (some possible datasets can be found here). Download two satellite image datasets for SS and decide which is the Source Dataset and which is the Target Dataset. Then, define an adequate class mapping between the two. Finally, repeat some of the previous steps with the selected datasets.

- [***********] Your own ideas!

DELIVERABLES

Once completed all the steps, you need to submit:

- PyTorch scripts with code for all the steps. Best if you:
  - provide your code in a public GitHub repository (if you want to keep it private, please add me as a collaborator)
  - provide a short documentation for your methods and functions, and describe your code with comments
  - provide a `README.md` file where you briefly describe how to setup and run experiments with your code
  - provide checkpoints and a module to easily load them and test, to prove the results you wrote on the report
    - You can upload the checkpoints outside the repository, but be sure that we will be able to download them
- A complete PDF report (paper-style). The report should contain the following sections: a brief introduction, description of the main related works, a methodological section describing the used algorithms and their purposes, an experimental section with all the results and discussions, and a final brief conclusion. Follow this link to open and create the template for the report

EXAMPLES OF QUESTIONS

- What is Semantic Segmentation?
- What is Federated Learning?
- What is Domain Generalization?
- What is Domain Adaptation?
- What is Source Free Domain Adaptation?
- What is Federated Source Free Domain Adaptation?
- Can you justify this result?
- Describe the network you used
- What is Statistical Heterogeneity?
- How do pseudo-labels work?
- How does FDA work?

REFERENCES

[1] Survey on SS: A Brief Survey on Semantic Segmentation with Deep Learning
[2] BiSeNet V2 network: BiSeNet V2: Bilateral Network with Guided Aggregation for Real-time Semantic Segmentation
[3] IDDA dataset: IDDA: a large-scale multi-domain dataset for autonomous driving
[4] GTA5 dataset: Playing for Data: Ground Truth from Computer Games
[5] SS metrics: Metrics to Evaluate your Semantic Segmentation Model
[6] Federated Averaging (FedAvg): Communication-efficient learning of deep networks from decentralized data
[7] Survey on FL: Federated learning: Challenges, methods, and future directions
[8] FedDrive: FedDrive: Generalizing Federated Learning to Semantic Segmentation in Autonomous Driving

[9] Survey on DA: A Review of Single-Source Deep Unsupervised Visual Domain Adaptation
[10] Source-Free Domain Adaptation for Semantic Segmentation
[11] LADD: Learning Across Domains and Devices: Style-Driven Source-Free Domain Adaptation in Clustered Federated Learning
[12] FDA: FDA: Fourier Domain Adaptation for Semantic Segmentation
[13] Pseudo-labels: Bidirectional Learning for Domain Adaptation of Semantic Segmentation
[14] FedProx: Federated Optimization in Heterogeneous Networks
[15] SCAFFOLD: SCAFFOLD: Stochastic Controlled Averaging for Federated Learning
[16] Server optimizers: Adaptive Federated Optimization