Take home assignment
Federated Learning for Image Classification

Elizaveta Uvarova

September 2024

1 Introduction

The task was to develop a simple federated learning system to train an image classification model collaboratively across multiple clients without sharing raw data between clients and the server. The code is provided in the form of a Google Colab notebook. It has a convolutional neural network (CNN) with 5 Independent and Identically Distributed (IID) clients. The code downloads the training and test sets directly to clients and runs for 7 epochs in 5 simulation rounds. After each round, the model's performance is evaluated on a test dataset that is loaded directly to the server, thus ensuring no data is shared between clients and the server. At the end of the simulation, a summary of each round's performance is printed and a confusion matrix of the model's performance is plotted. The highest centralized accuracy reached by this model was 63% and the highest weighted average accuracy was 0.63%. There is a possibility to add a differential privacy strategy to the model, however, this function has not yet been properly tuned and thus hinders significantly the learning of the model.

In this document, I will go over shortly the functions and logic of the code. I will also show the results from different runs and introduce a CNN model that could have provided better results but was not implemented due to the computational restrictions of the Google Colab notebook.

2 Methods

In this section, I will list the needed libraries and describe logic of the code, i.e., what do the customizable parameters in the third cell of the notebook do. Please note that the detailed descriptions of the functions can be found in the code itself, as it is more convenient to have them next to eachother.

2.1 Libraries

The simulation requires the following main libraries:

- Numpy
- scikit-learn
- PyTorch
- Flower

If the simulation is run in Google Colab notebook (as intended), the only installations required are as follows:

- flwr
- flwr[simulation]
- flwr_datasets

These can be installed directly in the notebook cell, as shown in the first cell of the code. At the time of making this documentation the `Flower` version used in the code was 1.11.1 and the `Python` version used was 2.4.0

2.2 Customization of the code

The code provides seven customization parameters. The user can choose the number of epochs and rounds for the model's training by adjusting the `ROUNDS` and `EPOCHS` parameters. The recommended number for rounds is 5 and for epochs is 7. The `NUM_PARTITIONS` parameter defines the number of clients. Simulation with 5 clients runs in about 20 minutes; simulation with 10 clients runs in about 35 minutes. If `DIFFERENTIAL_PRIVACY` is set to `True`, it will implement the differential privacy strategy. If `CONFUSION_MATRIX` is set to `True`, the code will plot a confusion matrix

for each evaluation round. If `VERBOSE` is set to `True` the code will print the training loss and accuracies of the clients. If the code is run on some other platform than Google Colab notebook, one can try to allocate more computational resources by choosing a different processor under the `DEVICE` parameter. If the code is run in Colab, it is recommended to use `CPU`.

3 Results

In this section, I will describe the results of three different simulations and provide the confusion matrices of their rounds. For all three simulations I kept the number of rounds as 5 and the number of epochs as 7, and only changed number of clients and whether to implement differential privacy. The three simulations are as follows: (1) 5 clients without differential privacy, (2) 10 clients without differential privacy, and (3) 5 clients with the implementation of differential privacy. In this section I will also describe a CNN model that I believe would have provided better results, but which I could not test with the federated learning due to computational restrictions.

The summary of the results are shown in the table below.

Table 1: Summaries of evaluation results of all three simulation. DP stands for differential privacy, C_A stands for centralized evaluation accuracy, and W_A stands for weighted average accuracy. The second row shows how long did it take to run the simulation. The rest of the table shows the accuracies for each round, where round 0 is the initial state before any training was done.

	5 clients, no DP		10 clients, no DP		5 clients, with DP	
Time to run	18 min		35 min		35 min	
	C_A	W_A	C_A	W_A	C_A	W_A
Round 0	0.09	-	0.08	-	0.10	-
Round 1	0.48	0.47	0.47	0.47	0.10	0.10
Round 2	0.58	0.57	0.58	0.59	0.08	0.08
Round 3	0.60	0.60	0.61	0.62	0.10	0.10
Round 4	0.61	0.61	0.63	0.63	0.08	0.09
Round 5	0.63	0.62	0.63	0.63	0.11	0.11

3.1 5 clients without differential privacy

Figure 1. shows the confusion matrices for the simulation that runs with the code as it is in the notebook. It will do 5 rounds of 7 epochs for 5 IID clients without the differential privacy. The simulation takes approximately 20 minutes to run and achieves a centralized evaluation accuracy of 63% and a weighted average accuracy of 62% by round 5.

3.2 10 clients without differential privacy

Figure 2. shows the confusion matrices for the simulation that runs 5 rounds of 7 epochs for 10 IID clients without the differential privacy. The simulation takes approximately 35 minutes to run and achieves a centralized evaluation accuracy of 63% and a weighted average accuracy of 63% by round 4.

3.3 5 clients with differential privacy

Figure 3. shows the confusion matrices for the simulation that runs 5 rounds of 7 epochs for 5 IID clients and then implements the differential privacy. The simulation takes approximately 35 minutes to run. This model has not yet been fully developed, and thus does not achieve any significant results over the rounds.

3.4 Deeper CNN model

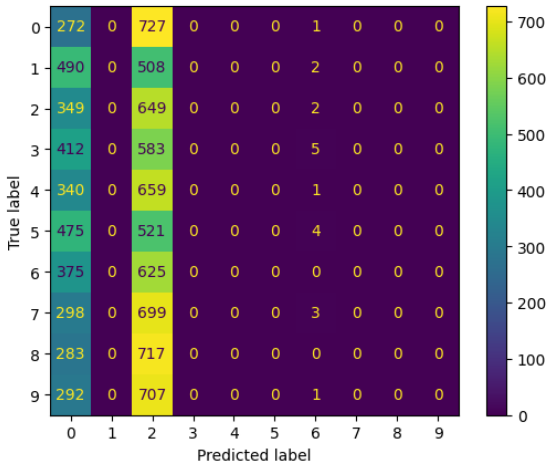
This is an experimental model that I have not implemented into the federated learning due to the computational costs. I have tested this model without the federated learning and it performs a lot better than the model used in the task. However, this model is also a lot more computationally heavy, as it has significantly more layers and neurons, and only pools every second layer, instead of every other layer. Even without the federated learning, it takes almost 1.5h to run seven epochs, and every time it has crushed either my laptop or my internet connection. Thus, I have not had the chance to test this model on client simulation. However, if I had a better equipment, e.g. CSC's supercomputer, this would be the model I would try to implement for this task.

```
class Net(nn.Module):
    def __init__(self) -> None:
        super(Net, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, 3, padding=1)
        self.conv2 = nn.Conv2d(32, 64, 3, padding=1)
        self.conv3 = nn.Conv2d(64, 128, 3, padding=1)
        self.conv4 = nn.Conv2d(128, 128, 3, padding=1)
        self.conv5 = nn.Conv2d(128, 256, 3, padding=1)
        self.conv6 = nn.Conv2d(256, 256, 3, padding=1)
        self.pool = nn.MaxPool2d(2, 2)
        self.fc1 = nn.Linear(256 * 4 * 4, 1024)
        self.fc2 = nn.Linear(1024, 512)
```

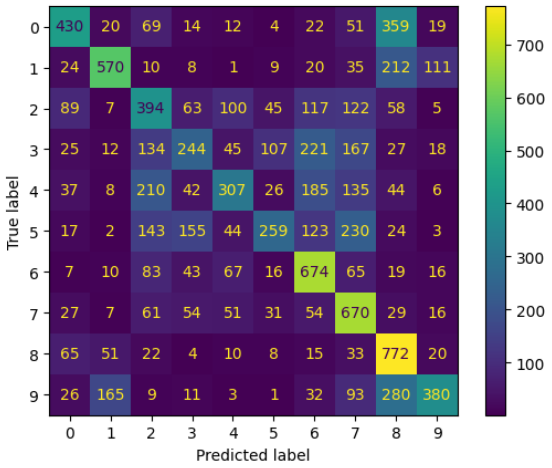
```
        self.fc3 = nn.Linear(512, 10)

    def forward(self, x: torch.Tensor) -> torch.Tensor:
        x = F.relu(self.conv1(x))
        x = F.relu(self.conv2(x))
        x = self.pool(x)
        x = F.relu(self.conv3(x))
        x = F.relu(self.conv4(x))
        x = self.pool(x)
        x = F.relu(self.conv5(x))
        x = F.relu(self.conv6(x))
        x = self.pool(x)

        x = x.view(-1, 256 * 4 * 4)
        x = F.relu(self.fc1(x))
        x = F.relu(self.fc2(x))
        x = self.fc3(x)
        return x
```

Figure 1: Confusion matrices for each round with five clients without differential privacy. C_A stands for centralized evaluation accuracy and W_A stands for weighted average accuracy.

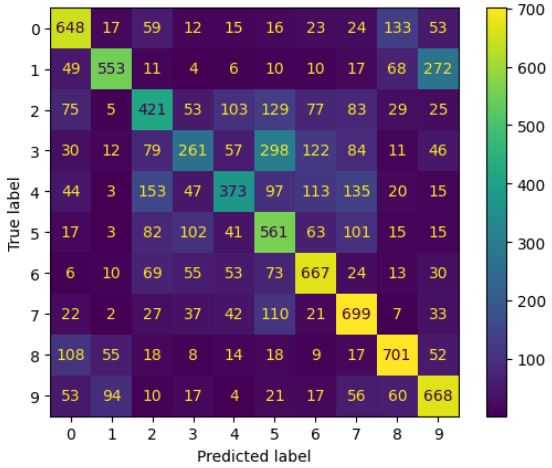
(a) Round 0, $C_A = 0.09$, $W_A =$ None



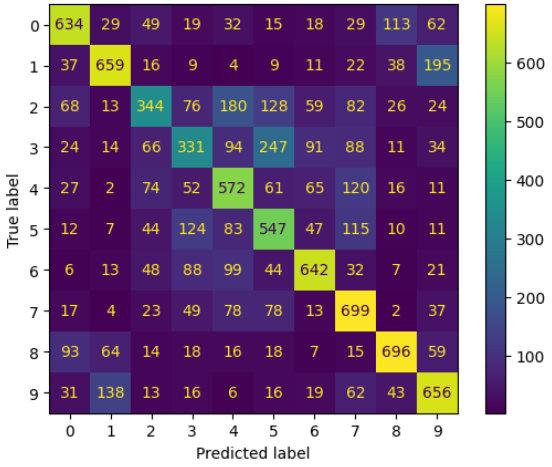
(b) Round 1, $C_A = 0.48$, $W_A = 0.47$



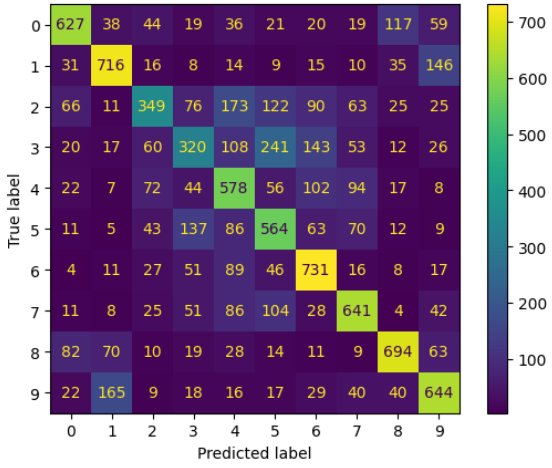
(c) Round 2, $C_A = 0.58$, $W_A = 0.57$



(d) Round 3, $C_A = 0.60$, $W_A = 0.60$



(e) Round 4, $C_A = 0.61$, $W_A = 0.61$



(f) Round 5, $C_A = 0.63$, $W_A = 0.62$

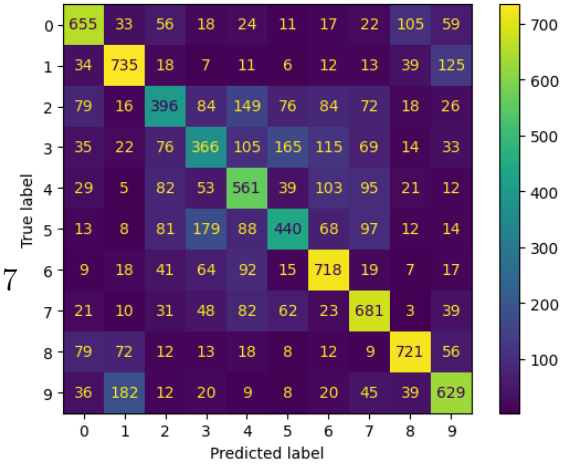
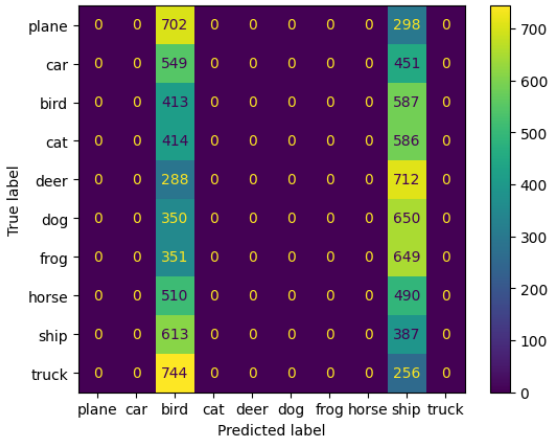
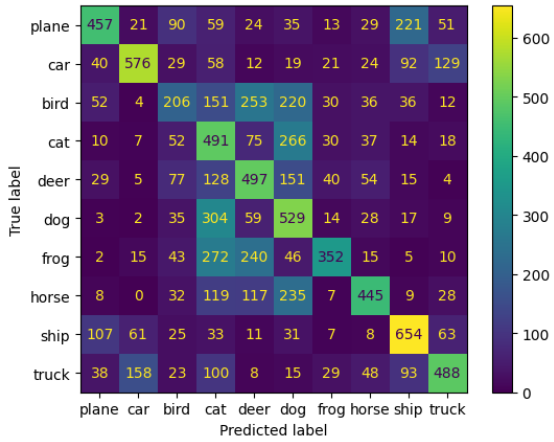


Figure 2: Confusion matrices for each round with ten clients without differential privacy. C_A stands for centralized evaluation accuracy and W_A stands for weighted average accuracy.

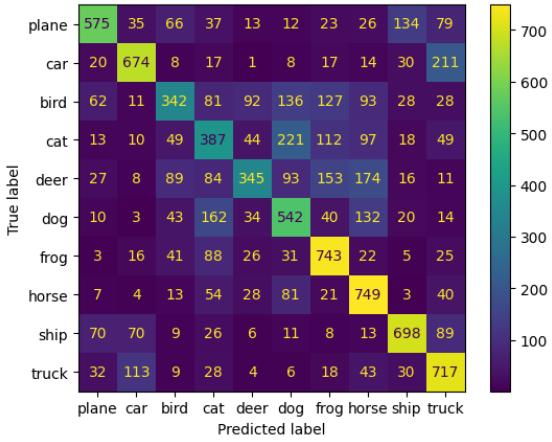
(a) Round 0, $C_A = 0.08$, $W_A =$ None



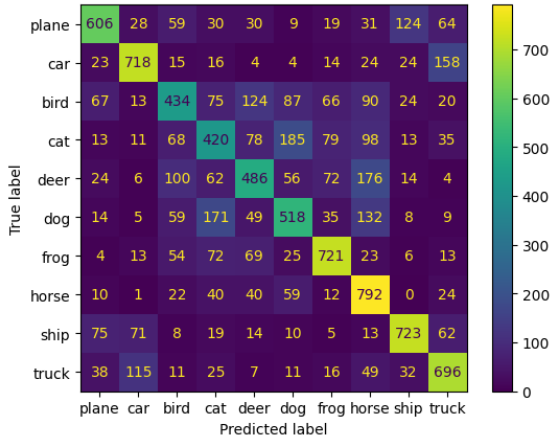
(b) Round 1, $C_A = 0.47$, $W_A = 0.47$



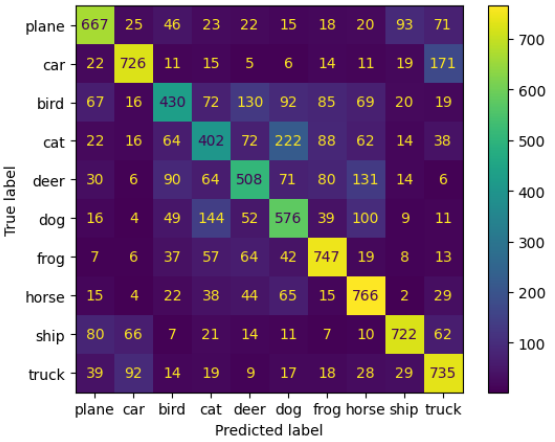
(c) Round 2, $C_A = 0.58$, $W_A = 0.59$



(d) Round 3, $C_A = 0.61$, $W_A = 0.62$



(e) Round 4, $C_A = 0.63$, $W_A = 0.63$



(f) Round 5, $C_A = 0.63$, $W_A = 0.63$

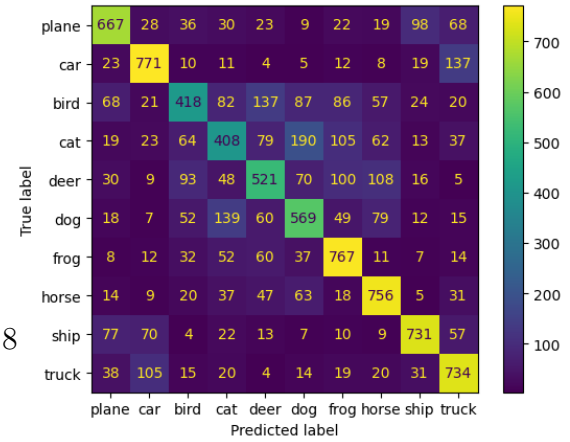
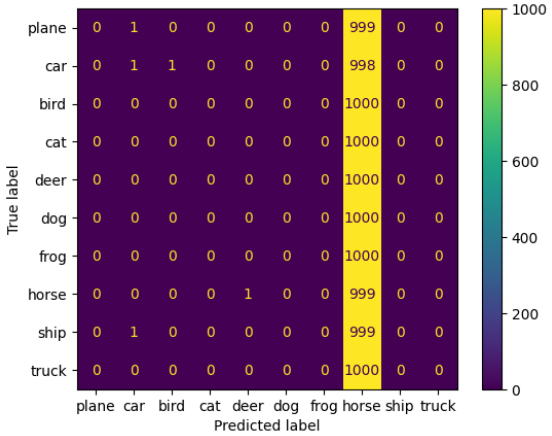
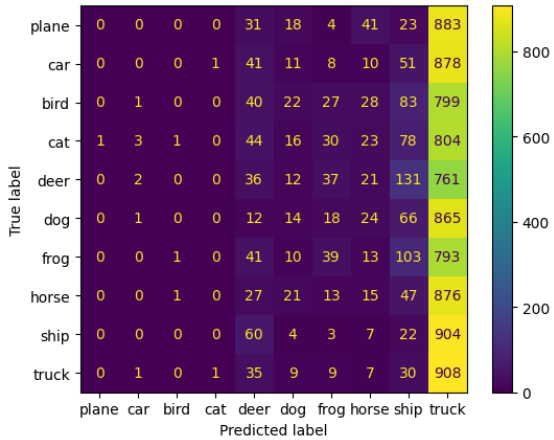


Figure 3: Confusion matrices for each round with 5 clients and implementing differential privacy. C_A stands for centralized evaluation accuracy and W_A stands for weighted average accuracy.

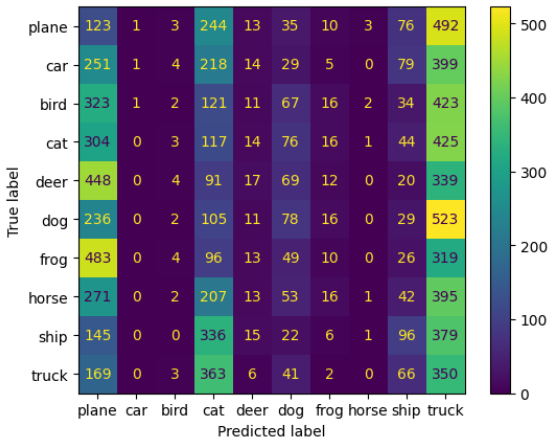
(a) Round 0, $C_A = 0.10$, $W_A =$ None



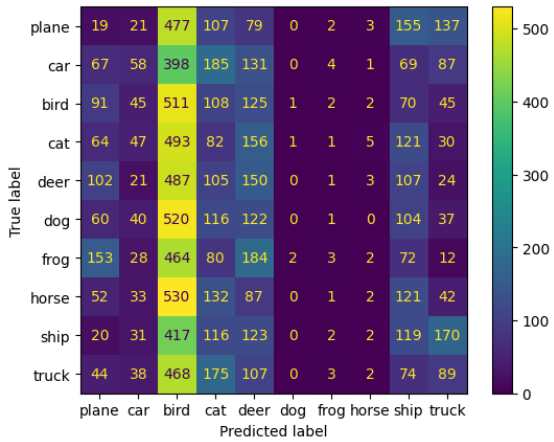
(b) Round 1, $C_A = 0.10$, $W_A = 0.10$



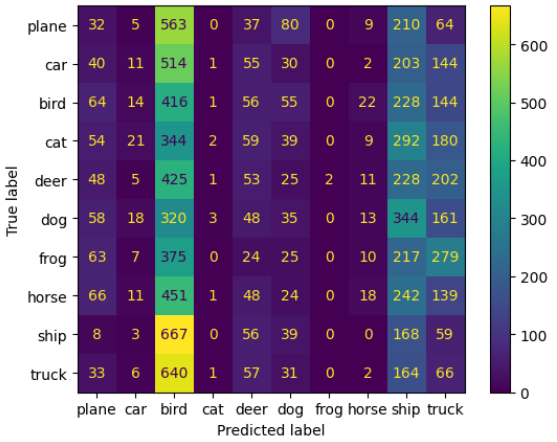
(c) Round 2, $C_A = 0.08$, $W_A = 0.08$



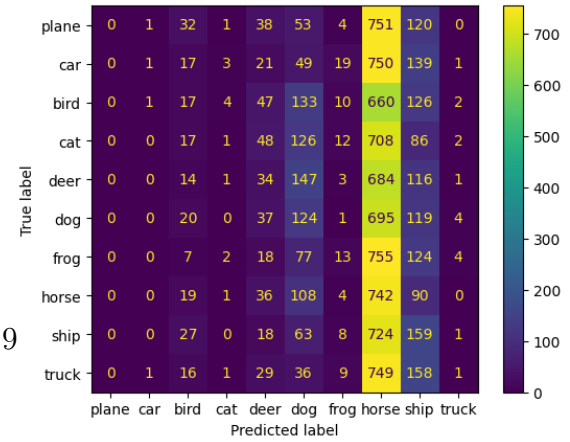
(d) Round 3, $C_A = 0.10$, $W_A = 0.10$



(e) Round 4, $C_A = 0.08$, $W_A = 0.09$



(f) Round 5, $C_A = 0.11$, $W_A = 0.11$



4 Conclusions

In this task I have developed a simple federated learning system to train an image classification model collaboratively across multiple IID clients without sharing raw data between clients and the server. I have simulated the framework on 5 and 10 clients, with each client having its own local dataset partitioned from the overall dataset. Each client trained the model on its local data for 7 epochs and the simulation ran for 5 rounds. After each communication round the global model's performance was evaluated on a separate test dataset, and accuracy and loss were tracked over communication rounds. The highest centralized accuracy reached by this model was 63% and the highest weighted average accuracy was 0.63%. I have attempted to implement a central differential privacy approach, however, I did not have enough time to fully develop it.

I have also provided a deeper and better performing CNN model, that could not be implemented in the federated learning strategy due to computational costs. The source code is provided as a Google Colab notebook with necessary comments and instruction on how to run the code.

References

[1] Flower Framework Differential Privacy. `https://flower.ai/docs/framework/how-to-use-differential-privacy.html`. [Accessed 15-09-2024].

[2] Flower Framework Tutorial. `https://flower.ai/docs/framework/tutorial-series-use-a-federated-learning-strategy-pytorch.html`. [Accessed 15-09-2024].