Assignment 4

Group number: 12

Students' Names:

Ahmed Ahmed Alwasefy

Ibrahim Mostafa Elshenhapy

Mohamed Salah Gabr

Date:

2023/07/08

Task 4

1- Modifications in code

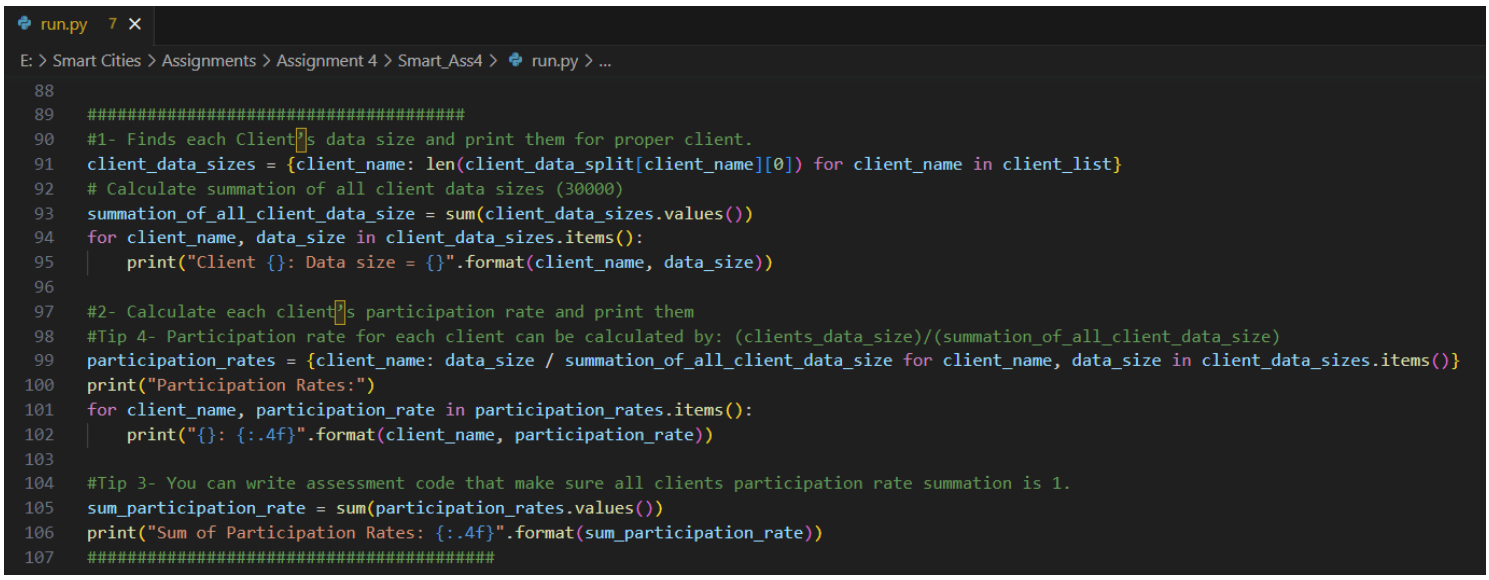
1. Please provide images of modifications in code. Highlight the parts you have changed.

File run.py modifications

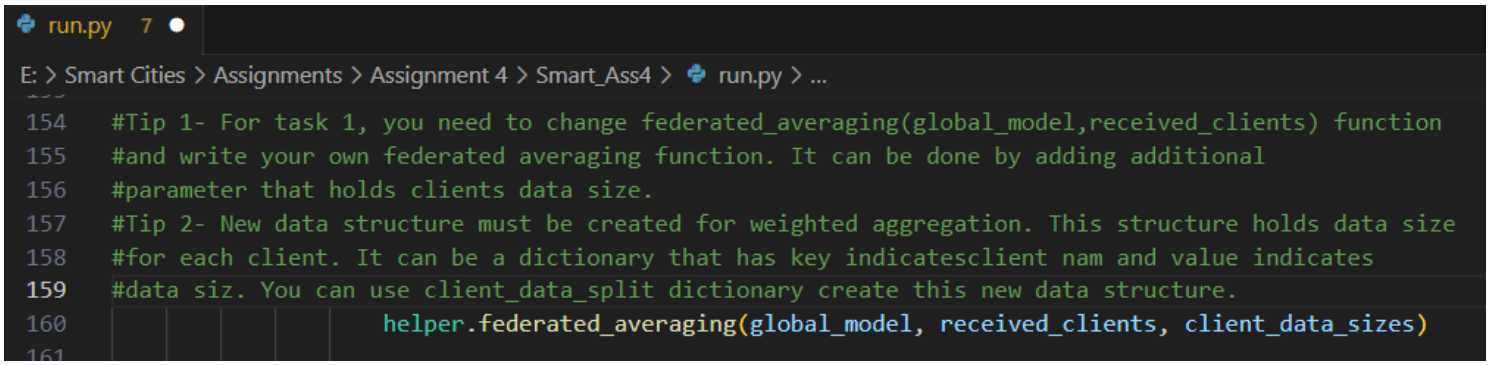
Q1 and Q2 Code modifications (run.py)

1- Making weighted aggregation with these participation rates

2- Calculate each client's participation rate and print them



Q3 Code modifications (run.py)



File helper.py modifications

Q3 Code modifications (helper.py)

3- Making weighted aggregation with these participation rates.

helper.py 9+

E: > Smart Cities > Assignments > Assignment 4 > Smart_Ass4 > helper.py > federated_averaging

```python
#Tip 1- For task 1, you need to change federated_averaging(global_model,received_clients) function
#and write your own federated averaging function. It can be done by adding additional
#parameter that holds clients data size.
#Tip 2- New data structure must be created for weighted aggregation. This structure holds data size
#for each client. It can be a dictionary that has key indicates client name and value indicates
#data size. You can use client_data_split dictionary create this new data structure.
################################
def federated_averaging(server, clients, client_data_sizes):
    target = {name: value.to(device) for name, value in server.named_parameters()}
    sources = []
    data_sizes = []
    #Loop in recived clients
    for client_name, client_model in clients.items():
        source = {name: value.to(device) for name, value in client_model.named_parameters()}
        sources.append(source)
        #Calculate data_size again but this time save it as tensor
        #Tried not to repeat this steps but i had some errors and this already working
        data_size = client_data_sizes[client_name] * torch.ones(1).to(device)
        data_sizes.append(data_size)

    # Calculate the summation of data size (30000)
    total_data_size = sum(client_data_sizes.values())

    # Perform weighted aggregation
    for name in target:
        # Just loop again to apply the weighted aggregation
        # by multiply with the weight of data which is (data_size / total_data_size)
        weighted_aggregation = torch.stack(
            [source[name].data * (data_size / total_data_size)
            for source, data_size in zip(sources, data_sizes)])
        target[name].data = torch.sum(weighted_aggregation , dim=0).clone()
################################
```

Q4 Code modifications (helper.py)

4- Creating CNN architecture.

helper.py 9+

E: > Smart Cities > Assignments > Assignment 4 > Smart_Ass4 > helper.py > federated_averaging

```python
# MODEL CREATION
##################
#Creating CNN architecture.
class Net(nn.Module):
    def __init__(self, num_class, dim):
        super(Net, self).__init__()
        #First CNN Layer 1 -> 32 units
        self.conv1 = nn.Conv2d(1, 32, kernel_size=3, stride=1, padding=1)
        #Second CNN layer 32 -> 64 units
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3, stride=1, padding=1)
        #The 64 unit need to be flatted first (old)
        # So the 64 * dim[0] divided by 4 and make it floor (use // instead of /)
        self.flatten_size = 64 * (dim[0]//4) * (dim[1]//4)
        #First dense layer (Fully Connected) 64 -> 128
        self.fc1 = nn.Linear(self.flatten_size, 128)
        #Second and last dense layer will take 128 unit to number of classes
        #128 -> len(classes)
        self.fc2 = nn.Linear(128, num_class)

    def forward(self, x):
        # Apply the first convolutional layer (conv1) with ReLU activation.
        x = F.relu(self.conv1(x))
        # Perform max pooling with a kernel size of 2 and stride of 2.
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        # Apply the second convolutional layer (conv2) with ReLU activation.
        x = F.relu(self.conv2(x))
        # Perform max pooling with a kernel size of 2 and stride of 2.
        x = F.max_pool2d(x, kernel_size=2, stride=2)
        # Reshape the tensor by flattening it. (old)
        x = x.view(-1, self.flatten_size)
        # Apply the first fully connected layer (fc1) with ReLU activation. (old)
        x = F.relu(self.fc1(x))
        # Apply the second and final fully connected layer (fc2).
        x = self.fc2(x)
        return x
######################
```

2- Report the assignment

In this assignment, we present our findings on the implementation of weighted aggregation in federated learning. We aimed to address the issue of varying client data sizes and their impact on model performance. By assigning participation rates to each client and incorporating them into the aggregation process, we sought to improve the overall accuracy of the model.

1- Client Data Size Calculation:

We iterated through the client list to determine the size of each client's data. We saved this information in a dictionary for future reference.

2- Participation Rate Calculation:

Next, we calculated the participation rate for each client. It was obtained by dividing the client's data size by the sum of all data sizes. We also verified the accuracy of the rates by summing them; they should ideally add up to 1.

3- Weighted Aggregation:

The participation rates obtained were utilized in the weighted aggregation of the model results. The aggregation process ensured that clients with larger data sizes had a proportionally greater impact on the model's outcome. This approach accounted for the data-driven nature of deep learning and mitigated the negative impact of clients with limited data.

4- Model Architecture:

To evaluate the effectiveness of the weighted aggregation, we designed a deep learning model using a Convolutional Neural Network (CNN). The model consisted of two convolutional layers followed by two dense layers. We chose a moderate number of units for each layer to strike a balance between complexity and running time.

Results: We observed the following outcomes during our experimentation:

Accuracy Improvement: When comparing the aggregated results to those of non-aggregated data, we found a notable increase in accuracy. Initially, we trained the model on 10 clients with three rounds, resulting in an accuracy of 93.2%. However, after incorporating weighted aggregation, the accuracy improved to 96.1%. This significant boost indicates the effectiveness of the approach in enhancing model performance.

Impact of CNN Architecture: The implementation of a Convolutional Neural Network (CNN) architecture played a crucial role in further improving the model's accuracy. The CNN architecture allowed the model to learn more complex representations from the data, resulting in improved classification accuracy.

3- Results

Please provide your output (figures, texts, …):

1. A snapshot of the command line output

Q1

```
Creating Model...
Total Data Size:30000
Client client_1: Data size = 2250
Client client_2: Data size = 750
Client client_3: Data size = 2250
Client client_4: Data size = 750
Client client_5: Data size = 2250
Client client_6: Data size = 750
Client client_7: Data size = 2250
Client client_8: Data size = 750
Client client_9: Data size = 2250
Client client_10: Data size = 750
Client client_11: Data size = 2250
Client client_12: Data size = 750
Client client_13: Data size = 2250
Client client_14: Data size = 750
Client client_15: Data size = 2250
Client client_16: Data size = 750
Client client_17: Data size = 2250
Client client_18: Data size = 750
Client client_19: Data size = 2250
Client client_20: Data size = 750
```

Q2

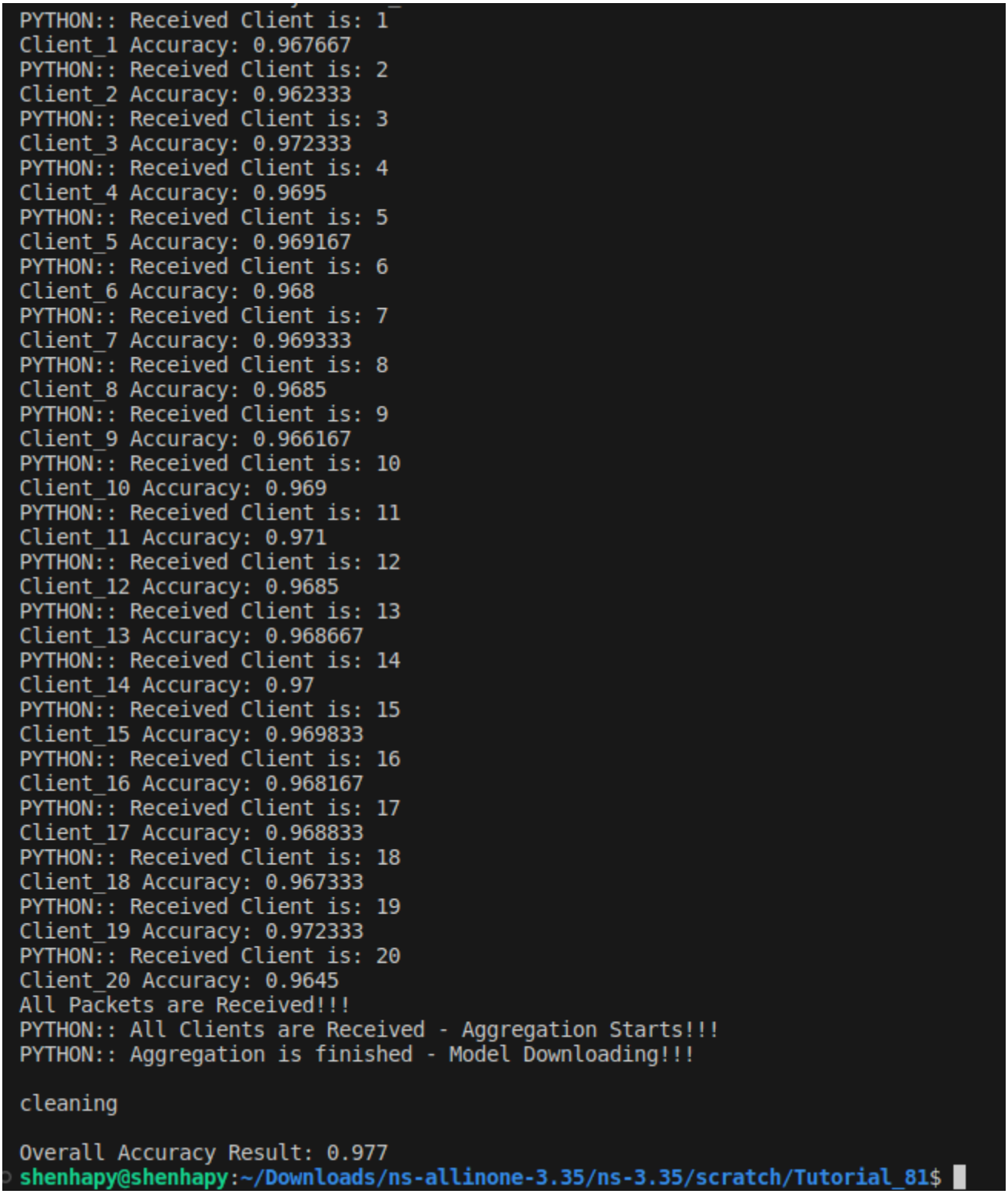
```
Participation Rates:
client_1: 0.0750
client_2: 0.0250
client_3: 0.0750
client_4: 0.0250
client_5: 0.0750
client_6: 0.0250
client_7: 0.0750
client_8: 0.0250
client_9: 0.0750
client_10: 0.0250
client_11: 0.0750
client_12: 0.0250
client_13: 0.0750
client_14: 0.0250
client_15: 0.0750
client_16: 0.0250
client_17: 0.0750
client_18: 0.0250
client_19: 0.0750
client_20: 0.0250
Sum of Participation Rates: 1.0000
```

2. A snapshot of the Model training and results

For the last Round (Round 9) [Total 10 Rounds 0 ~> 9]

```
Create nodes.
Assign IP Addresses.
Create sockets.
Run Simulation.
Server: 10.2.1.1
Client: 10.1.0.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.1.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.2.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.3.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.4.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.5.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.6.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.7.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.8.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.9.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.10.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.11.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.12.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.13.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.14.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.15.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.16.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.17.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.18.1 m_peer:03-07-0a:02:01:01:7d:11:00
Client: 10.1.19.1 m_peer:03-07-0a:02:01:01:7d:11:00
All Packets Are Sent by Client_1!!!
All Packets Are Sent by Client_2!!!
All Packets Are Sent by Client_3!!!
All Packets Are Sent by Client_4!!!
All Packets Are Sent by Client_5!!!
All Packets Are Sent by Client_6!!!
All Packets Are Sent by Client_7!!!
All Packets Are Sent by Client_8!!!
All Packets Are Sent by Client_9!!!
All Packets Are Sent by Client_10!!!
All Packets Are Sent by Client_11!!!
All Packets Are Sent by Client_12!!!
All Packets Are Sent by Client_13!!!
All Packets Are Sent by Client_14!!!
All Packets Are Sent by Client_15!!!
All Packets Are Sent by Client_16!!!
All Packets Are Sent by Client_17!!!
All Packets Are Sent by Client_18!!!
All Packets Are Sent by Client_19!!!
All Packets Are Sent by Client_20!!!
```

Each client accuracy and the final accuracy after 10 rounds.



Overall Accuracy Result is 0.977

3. Log for all the Overall Accuracy Result for the 10 rounds

Round 0: Overall Accuracy Result: 0.912167

Round 1: Overall Accuracy Result: 0.9495

Round 2: Overall Accuracy Result: 0.960167

Round 3: Overall Accuracy Result: 0.965167

Round 4: Overall Accuracy Result: 0.969

Round 5: Overall Accuracy Result: 0.9715

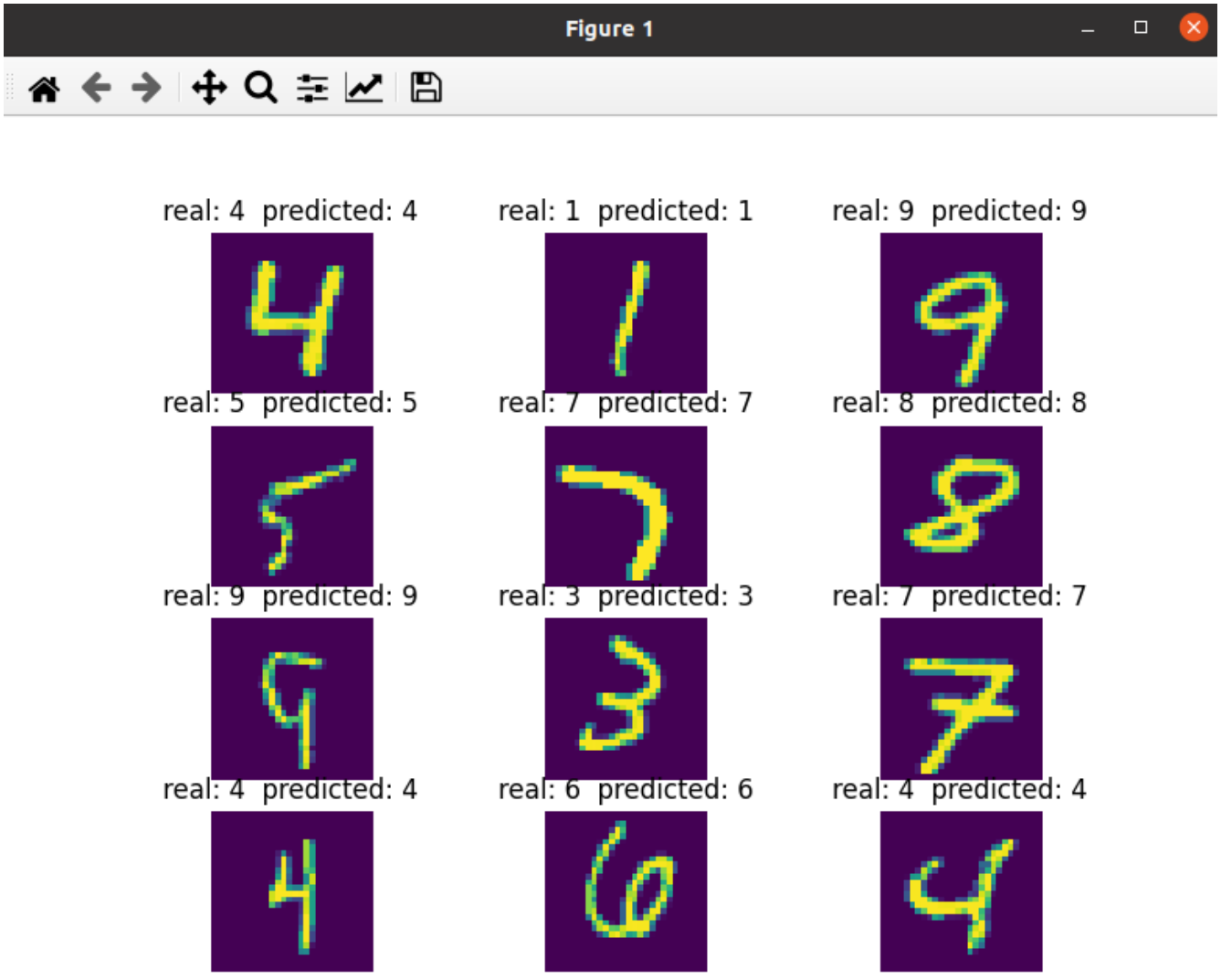
Round 6: Overall Accuracy Result: 0.973667

Round 7: Overall Accuracy Result: 0.975167

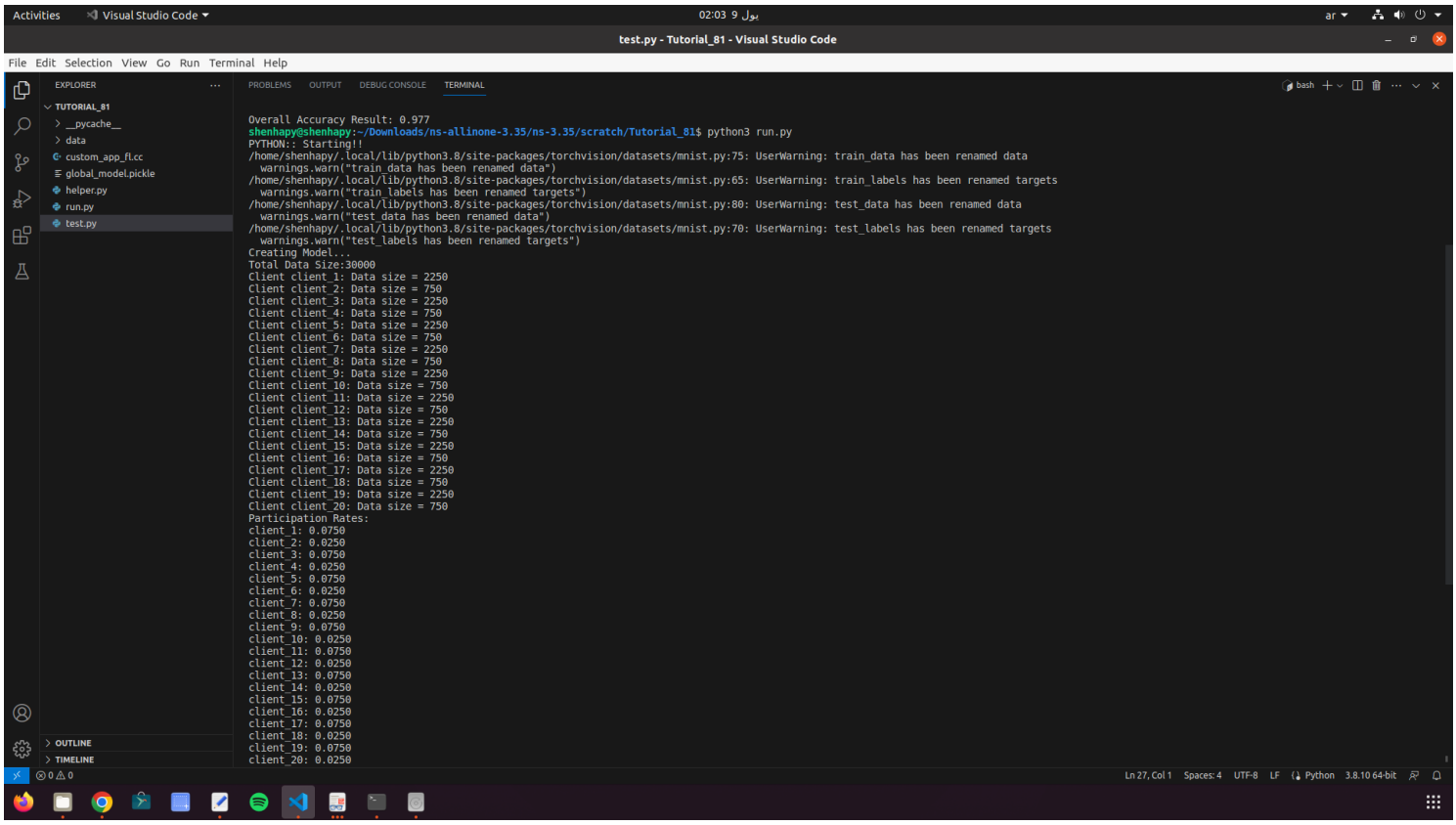
Round 8: Overall Accuracy Result: 0.976333

Round 9: Overall Accuracy Result: 0.977

4. Test Results (test.py)



5. Whole Screen screenshot



4-List of files

Please mention the name of the files you have attached in Brightspace. Also, attach this file and modified code file in Brightspace.

1-run_GP12.py
1-helper_GP12.py
1-test_GP12.py

2-Report_GP12.pdf

global_model.pickle

custom_app_fl.cc