CMPE 492

Federated Learning for Sleep Quality Prediction with Wearable Devices

Berkant Koç

Advisor: Özlem Durmaz İncel

TABLE OF CONTENTS

1. INTRODUCTION

The rise of wearable devices, such as smartwatches, fitness trackers, and medical sensors, has led to an explosion of data that can be used to understand human behavior and health better. These devices collect a wide range of data, including heart rate, steps taken, sleep patterns, and even blood glucose levels. We can gain insights into individual health status and predict potential health issues by analysing this data using deep learning techniques.

However, the privacy concerns associated with wearable data pose a significant challenge for traditional centralized machine-learning approaches. These approaches typically require data to be collected and stored in a central location for analysis. This raises concerns about the security of sensitive health information and the potential for data breaches or misuse.

Federated learning, a novel approach to machine learning, offers a solution to this problem. This technique enables collaborative learning across multiple devices without compromising user privacy. In a federated learning setting, each device holds its data locally, and the model training occurs in a decentralized manner without the need for data to be transferred to a central server. The central server only aggregates the model updates from the devices, and the trained model is then shared back to the devices.

Federated learning has gained attention recently for its potential to improve healthcare outcomes while preserving data privacy. In this context, federated learning has emerged as a promising approach for improving health outcomes by training deep learning models on distributed wearable data. This approach could revolutionize healthcare by enabling accurate and personalized predictions for individuals while maintaining their privacy.

1.1 Applications of Federated Learning on Wearable Data

Federated learning has several advantages over traditional machine learning approaches, especially regarding wearable data. Firstly, it addresses the challenge of data heterogeneity across different devices, which is common in the wearable domain. By training models locally on each device, federated learning can account for device-specific data variations, leading to improved model performance.

Secondly, federated learning allows for personalized model training, which is crucial in healthcare applications. Each individual's health data is unique, and a personalized model trained on their data can provide more accurate predictions than a model trained on aggregated data. Federated learning enables the training of personalized models while preserving data privacy.

Thirdly, federated learning can overcome the challenge of data imbalance. Wearable data is often imbalanced, with certain health conditions or behaviors being underrepresented. Federated learning can address this by training models on distributed data across different devices, enabling the inclusion of more diverse data.

Federated learning has already shown promising results in several healthcare applications. For example, a study used federated learning to train a deep learning model for detecting atrial fibrillation (a heart condition) on wearable data. The results showed that the federated learning approach achieved comparable performance to a centralized learning approach while preserving data privacy.

1.2 Ethical Considerations on Federated Learning

Federated learning is a machine learning approach where an algorithm is trained across multiple decentralized edge devices or servers holding local data samples without exchanging the data itself. This has many potential applications and advantages, especially regarding data privacy and security. However, as with any technology, important ethical considerations should be taken into account:

1. **Data Privacy**: Even though federated learning doesn't require sharing raw data,

there could still be indirect privacy issues. For example, if the model is learning from sensitive data (like personal medical records), it could infer some information about the individual data entries. Steps should be taken to anonymize the data further or add additional layers of security to protect privacy.

2. **Security and Data Integrity**: Federated learning involves communication between different nodes, which can open up opportunities for cybersecurity attacks. For instance, an attacker could try to manipulate the learning process by influencing the updates sent from a certain node. Also, the privacy-preserving methods used could be reverse-engineered.
3. **Bias and Fairness**: Since federated learning relies on data from multiple nodes, there can be issues if the data in different nodes is biased in some way. For instance, one node might have data that is skewed towards a particular demographic. This could cause the model to perform better for some groups than others, leading to unfair results.
4. **Data Ownership and Governance**: Federated learning raises questions about who owns the data and the models trained on it. It's important to establish clear policies and guidelines about data ownership, model ownership, and data usage.
5. **Transparency and Explainability**: Like other forms of machine learning, it can be challenging to understand how federated learning models make decisions, leading to potential ethical issues. For example, if a federated learning model makes a medical diagnosis or a financial decision, it's important to be able to understand how it arrived at that conclusion.
6. **Accessibility and Digital Divide**: If federated learning is being used to improve services or products, it's important to ensure that the benefits are being shared equitably. There is a risk that communities with less access to technology (and thus less data being contributed) might benefit less from federated learning.

2. PROJECT DEFINITION AND PLANNING

2.1 Project Definition

In our project, we want to predict people's sleep quality by applying state-of-the-art deep learning techniques and making it federated. We are using NetHealth open-source dataset, which was collected from 698 students in total during eight semesters. This comprehensive study includes data on communication patterns from mobile phones, sleep, and physical activity routines, students' family backgrounds, living conditions, personality, etc., from surveys. This study uses some parts of the basic survey and all the wearable device data. Therefore, preprocessing is applied to extract the required data to construct a sub-dataset for the analysis.

In our study, we intended to use Deep ConvLSTM architecture to analyze the wearable data. In short, the DeepConvLSTM model consists of convolutional layers of CNN and LSTM cells to memorize past data. We are inspired by the paper ***"Deep Convolutional and LSTM Recurrent Neural Networks for Multimodal Wearable Activity Recognition"*** by Francisco Javier Ordóñez and Daniel Roggen [1] . This paper uses the Deep ConvLSTM model to analyze wearable data and obtain heartwarming results. However, this paper did a classification task. Since we need sleep quality prediction, we must create a regression model. When we deeply analyzed this architecture, we decided that this model is unsuitable for applying our dataset. Therefore, we decided to compare conventional deep learning models' performance and the performance of federated learning.

We have five CNN-based deep learning models to predict sleep efficiency.

1. C-P-FC-FC
2. C-P-C-P-FC
3. C-C-P-C-C-P-FC-FC
4. C-P-C-P-FC-FC
5. C-P-C-P-C-P

- C represents the convolutional layer in CNN

- P represents the max pooling layer in CNN
- FC represents the fully connected layer in CNN

CNN stands for Convolutional Neural Network. It is a type of deep learning algorithm primarily used for image recognition, computer vision tasks, and other tasks involving analyzing grid-like structured data. CNNs are inspired by the organization of the visual cortex in animals.

The key idea behind CNNs is the use of convolutional layers. Convolution is a mathematical operation that involves applying a filter or kernel to an input image or data grid. The filter is small in size and slides over the entire input, performing element-wise multiplication and summing the results. This operation helps the network to learn local patterns or features from the input data.

CNNs typically consist of multiple layers, including convolutional, pooling, and fully connected layers. Convolutional layers extract features from the input data using filters. Pooling layers downsample the feature maps, reducing their spatial dimensions while retaining the vital information. Finally, fully connected layers combine the learned features and make predictions based on them.

During the training process, CNNs learn to automatically discover and extract meaningful features from the input data, allowing them to recognize patterns and objects in images effectively. The network is trained using labelled data, where the model adjusts its internal parameters through a process called backpropagation to minimize the difference between predicted and actual labels.

2.2 Project Planning

We divided our project into two parts; the first part is creating our deep learning model, and the second part is applying federated learning to our model.

In the first part, we decided which frameworks and deep learning architecture would be used in our project. We used the Flower [3] framework in the federated learning part and tensorflow for deep learning. After deciding on the framework, we started to look for feasible deep-learning architectures to use in our project. After a short literature review, we first decided to use Deep ConvLSTM architecture, but this model was incompatible with our dataset. We decided to use five simple CNN architectures to make the comparison.

In the second part of the project, we applied federated learning to our model. Since Flower is one of the most widely used frameworks in this area, we decided to go with Flower in this phase.

3. Related Works

Sleep Quality Prediction from Wearables using Convolution Neural Networks and Ensemble Learning [2] defines the models that we used in this study. In this [2] work, they investigated which models perform best for predicting sleep quality, and they have found random forest outperformed all the CNN-based models by using the same dataset as us. To apply federated learning, we chose these models. Therefore, this study can be considered as further work for this [2] study.

On the other hand, Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition [1] proposes a DeepConvLSTM for the activity recognition task using wearable devices. Since we are also interested in the data that are collected from wearable devices, this [1] work is also similar to our study.

4. Methods

4.1 Dataset

The data used in this study is gathered from the Net Health dataset [2], which is collected at Notre Dame University. This data covers eight semesters between Fall 2015 and Spring 2019. There are data on approximately 400 students from 2015 to 2017 and 300 from 2015 to 2019. Data collection comprises social networks, physical activity, sleep data, basic survey data, communication data, courses and grades data, and academic calendar data. Basic survey data include family background & demographic variables, student background, self-reported course, grade, major information, activities, clubs, musical tastes, personality - Big five, and social-psychological scales tests. Besides these, physical and psychological health-related questions, PSQI sleep quality test, political attributes and views, and computer, phone, and Fitbit device usage of students are included.

NetHealth dataset is a comprehensive, multi-year longitudinal dataset. So, some survey questions were only asked in some waves, Fig.1 [2] below shows the sleep efficiency in a given time scale. The missing information restricts this study from using every wave in deep learning training. This study focuses on Fitbit's sleep quality scores as our target value.

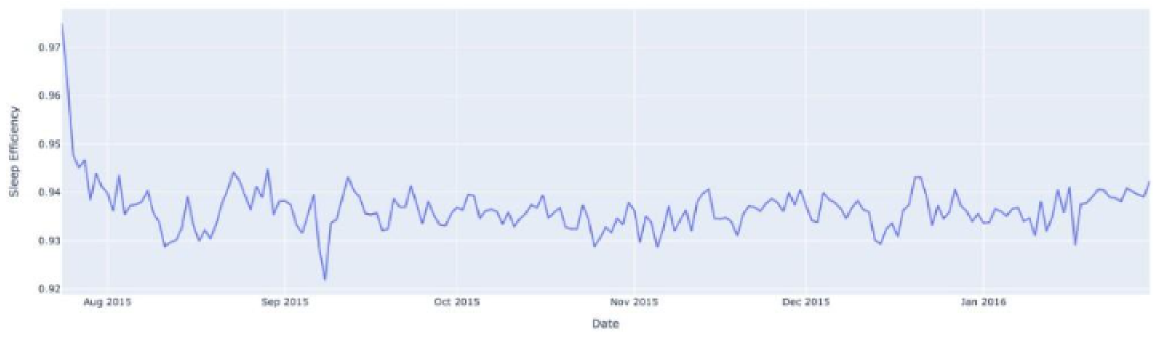


Figure 4.1 Sleep Efficiency for First and Second Semesters [2]

4.2 Preprocessing

As a target value in the scope of this study, we use the Fitbit sleep efficiency score collected from the wearable - Fitbit. The device uses its sensors to detect if someone is awake or asleep. Besides this information, total time in bed, awake time in bed, and sleeping time are also detected. The sleep efficiency score can be calculated thanks to these values, i.e., $minsasleep/(minsasleep + minsawake)$. The score is the ratio of asleep time in bed to the total time in bed and ranges between 0 and 1.

Before model evaluation, we concatenated all separated responses according to timestamps and participants. In addition, we converted the survey questions into numeric models. Besides, there were missing values in some survey responses for different semesters. In this study, we are concentrating on the first and second semester's data due to not being requested from the participants since the decided features have responses. Furthermore, students who still need to answer the questionnaires are subtracted. In the end, we have 200615 rows and 92 columns.

4.3 Model Details

In our project, we decided to use 5 different CNN models. These models are trained and tested in a previous study [2] in a centralized manner. We then created a simple federated learning architecture and ran these models separately.

Five CNN models are in these architectures:

- C-P-FC-FC
- C-P-C-P-FC
- C-C-P-C-C-P-FC-FC
- C-P-C-P-FC-FC
- C-P-C-P-C-P

Our federated learning phase consists of two steps. One is with 8 clients, and the other is with 4 clients.

5. Requirements Specification

5.1 Functional Requirements

1. Data Extraction: The system should be able to extract and preprocess data from the NetHealth open-source dataset, focusing on relevant data related to sleep patterns, physical activity, family backgrounds, living conditions, personality traits, and other relevant factors.
2. Deep Learning Model Training: The system must be able to train various CNN-based deep learning models, including the five different architectures you listed, using the preprocessed dataset.
3. Sleep Quality Prediction: The trained models should predict individuals' sleep quality based on the data input. The output should be a numerical value representing the sleep efficiency.
4. Federated Learning: The system must support a federated learning approach, wherein the model is trained across multiple decentralized devices holding local data samples, without data exchange.
5. Comparison: The system should compare the performance of the traditional deep learning models to the performance of the federated learning models.
6. Evaluation: The system needs to evaluate the models' performance using appropriate metrics such as Mean Absolute Error (MAE), Mean Squared Error (MSE), or others relevant for regression tasks.

5.2 Non-Functional Requirements

1. Accuracy: The system should provide accurate predictions of sleep quality.
2. Efficiency: The system should efficiently train models and make predictions within a reasonable time frame.
3. Scalability: The system should be scalable, capable of handling larger datasets or additional data fields in the future.
4. Security: The system must ensure privacy-preserving techniques, especially in a federated learning context, to protect sensitive information.
5. Usability: The system should be user-friendly and provide clear instructions or documentation for use.
6. Robustness: The system should be robust and handle different kinds of data without crashing or producing erroneous results.

5.3 System Requirements

1. Software: The system requires a deep learning framework like TensorFlow or PyTorch. It also needs a federated learning library such as Flower. It may require additional software for data preprocessing (e.g., pandas, numpy), data visualization (e.g., matplotlib, seaborn), and model evaluation.
2. Hardware: A powerful GPU for model training is essential. Depending on the data size, sufficient storage and memory are also necessary.
3. Network: Since federated learning is involved, a reliable internet connection is required to handle communication between different devices.
4. Operating System: It should work on major OS platforms like Windows, Linux, or macOS, depending on the development and deployment needs.
5. Libraries and Drivers: The system might require specific drivers for GPU acceleration (like CUDA for Nvidia GPUs) and specific versions of various software libraries for compatibility.

6. Design

6.1 Information Structure

The data used in this study is gathered from the Net Health dataset [2], which is collected at Notre Dame University. This data covers eight semesters between Fall 2015 and Spring 2019. There are data on approximately 400 students from 2015 to 2017 and 300 from 2015 to 2019. Data collection comprises social networks, physical activity, sleep data, basic survey data, communication data, courses and grades data, and academic calendar data. Basic survey data include family background & demographic variables, student background, self-reported course, grade, major information, activities, clubs, musical tastes, personality - Big five, and social-psychological scales tests. Besides these, physical and psychological health-related questions, PSQI sleep quality test, political attributes and views, and computer, phone, and Fitbit device usage of students are included.

6.2 Information Flow

In a classical Federated Learning task, the information flow occurs in this way:

1. Server sends initialization parameters to the client
2. Client runs the model with given parameters
3. After executing the model, client sends updated parameters to the server
4. Server does aggregation operation according to its strategy. For our study, we use FedAvg[4] strategy.
5. After aggregation, server sends these parameters to clients
6. Repeat 2-5

7. **Implementation**

In order to implement our model, we have used the TensorFlow library since it is highly compatible with the Flower framework [3], which will be used in the federated learning phase.

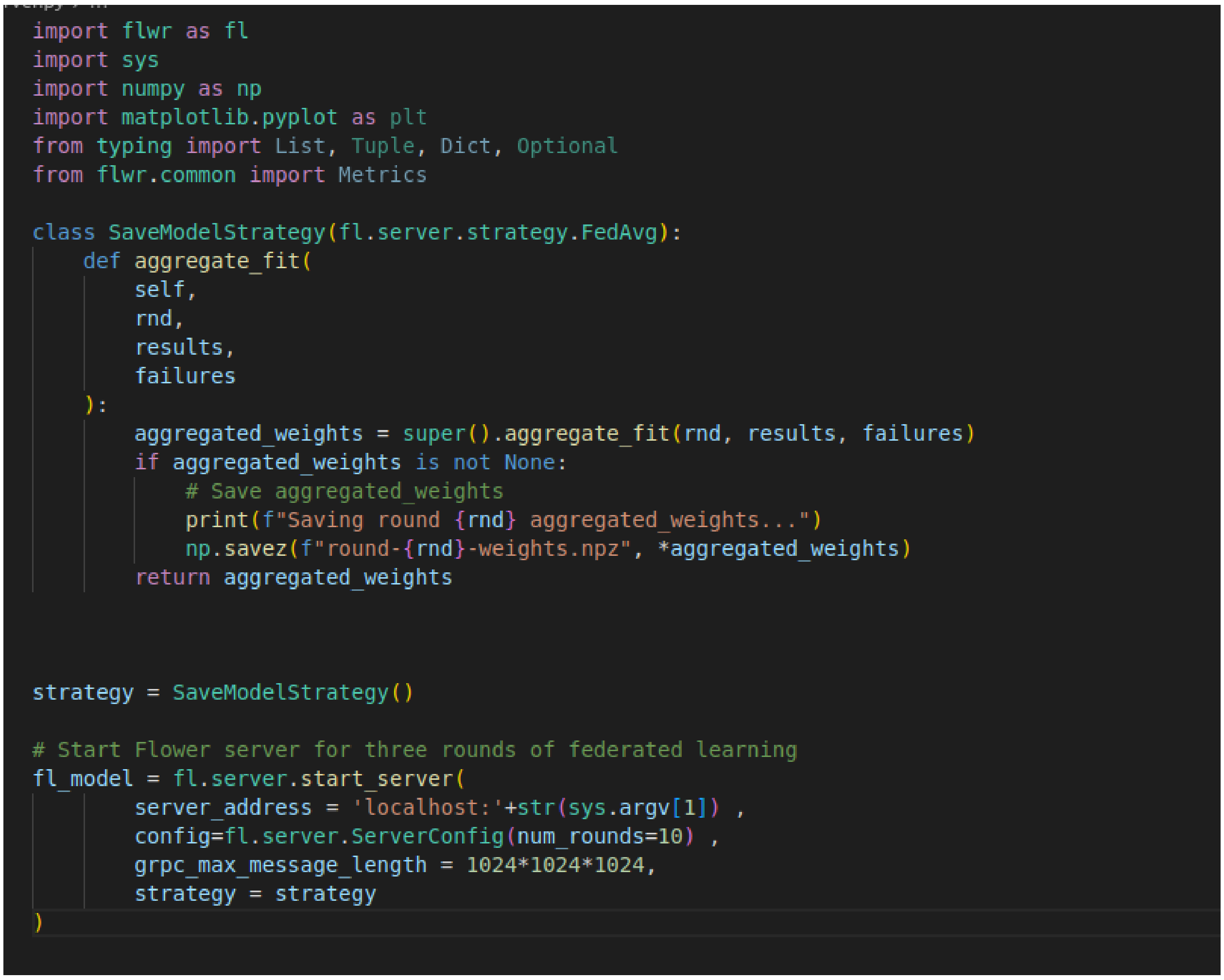
The Flower framework is an open-source research framework designed for federated learning (FL). It provides a comprehensive set of tools and functionalities to facilitate the development, experimentation, and evaluation of FL algorithms and systems. Flower aims to address the challenges associated with scaling FL experiments and accommodating the heterogeneity of edge devices.

Key features and capabilities of the Flower framework include:

1. Large-scale FL experiments: Flower offers facilities to execute large-scale FL experiments, allowing researchers to study FL algorithms with a significant number of clients.
2. Heterogeneous FL device scenarios: Flower supports the consideration of richly heterogeneous FL device scenarios. This means that the framework can accommodate devices with varying computational capabilities, network conditions, and other device-specific characteristics.
3. Seamless migration to real devices: Researchers can seamlessly migrate their experiments from simulated environments to real devices, enabling them to validate and evaluate their FL algorithms in real-world settings.
4. Scalability: Flower is designed to handle large-scale FL workloads efficiently. It can effectively execute FL experiments involving a considerable number of clients, even with resource-intensive requirements.

Here, I have published my repository. I have one server.py file and some clients.py files. They are explained in the following section.

7.1 Server



```python
import flwr as fl
import sys
import numpy as np
import matplotlib.pyplot as plt
from typing import List, Tuple, Dict, Optional
from flwr.common import Metrics

class SaveModelStrategy(fl.server.strategy.FedAvg):
    def aggregate_fit(
        self,
        rnd,
        results,
        failures
    ):
        aggregated_weights = super().aggregate_fit(rnd, results, failures)
        if aggregated_weights is not None:
            # Save aggregated_weights
            print(f"Saving round {rnd} aggregated_weights...")
            np.savez(f"round-{rnd}-weights.npz", *aggregated_weights)
        return aggregated_weights



strategy = SaveModelStrategy()

# Start Flower server for three rounds of federated learning
fl_model = fl.server.start_server(
        server_address = 'localhost:'+str(sys.argv[1]) ,
        config=fl.server.ServerConfig(num_rounds=10) ,
        grpc_max_message_length = 1024*1024*1024,
        strategy = strategy
)
```

Figure 7.1 Description of server.py file

My code, Figure 7.1, uses the Flower library to implement a federated learning server with a custom aggregation strategy SaveModelStrategy that extends the default FedAvg strategy. The custom strategy overrides the aggregate_fit method, which is called after each round of training to aggregate the model weights. In addition to performing the usual aggregation, it saves the aggregated weights to a file.

FedAvg, short for Federated Averaging [4] , is a common strategy used in federated learning for aggregating model updates. The idea is simple yet effective: each participating client (device) in the federated learning network trains a shared model locally on its data, and then sends the updates (changes in model parameters) to a central server. The server then averages these client updates to form a global model update.

The pseudocode for the FedAvg algorithm is as follows:

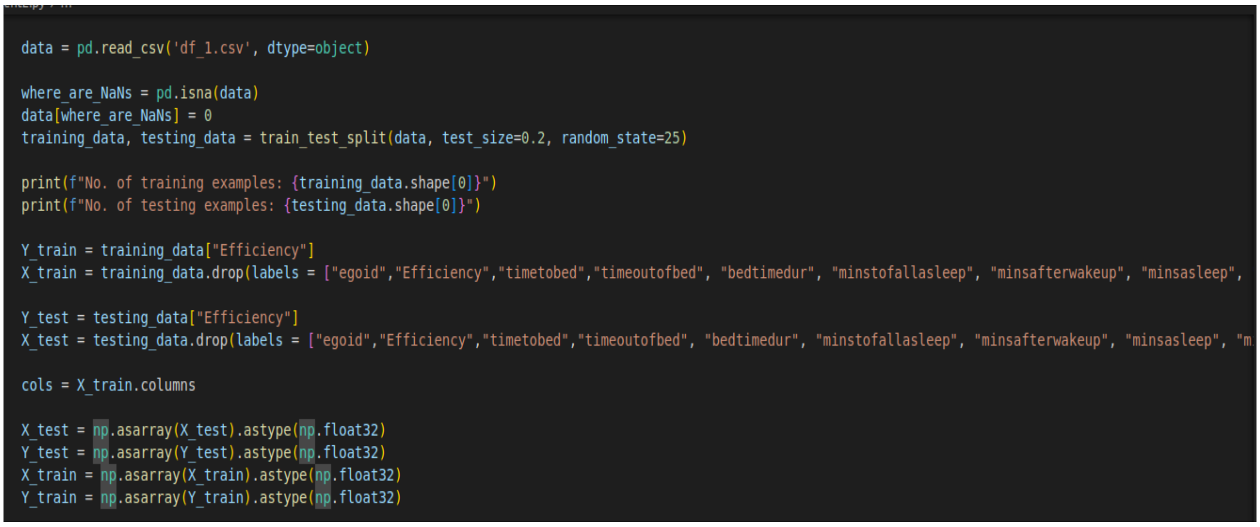
1. Initialize a global model at the server.
2. For each round, the server sends the global model to each participating client.
3. Each client trains the model locally using its own data and creates a local model.
4. Each client then sends their local model or model update to the server.
5. The server aggregates these models/updates by taking a weighted average, typically based on the number of samples from each client. This creates the new global model.
6. Repeat steps 2-5 for a certain number of communication rounds.

This strategy is a way to balance the local learning done by each client with global learning across the network. It is a simple and effective strategy, but it assumes that all clients have similar data distributions, which isn't always the case in real-world federated learning scenarios.

7.2 Clients

In federated learning, we need clients accompanied by a server. The number of clients can vary depending on the architecture of the system.

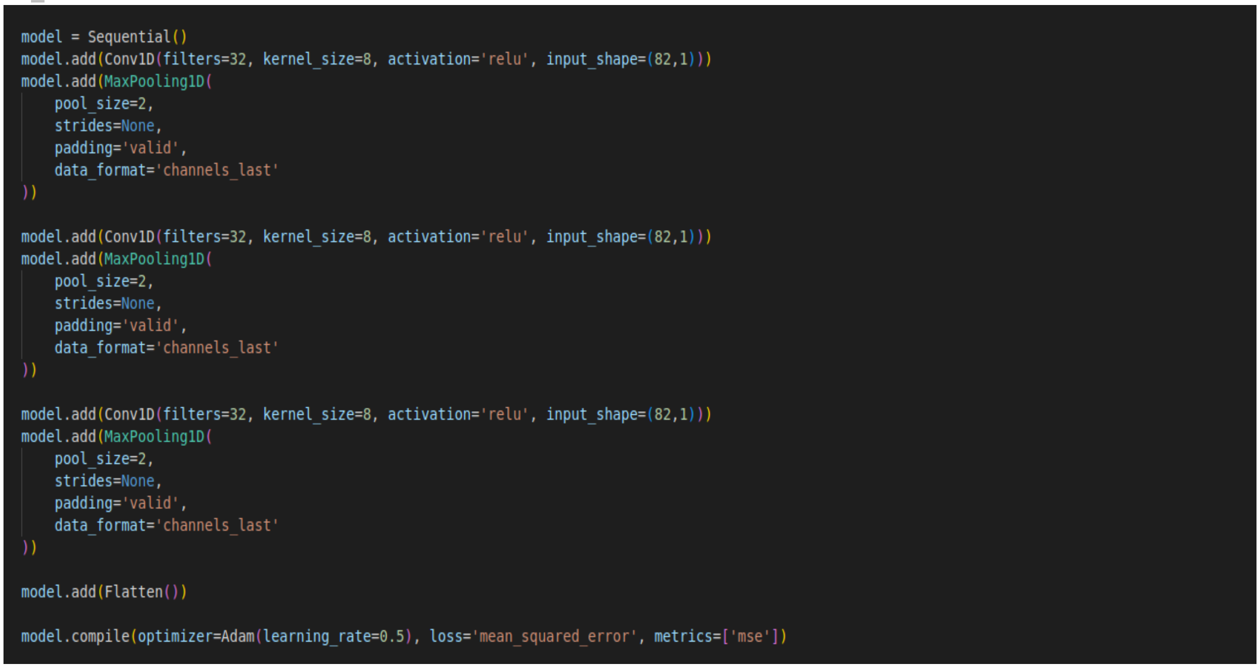
In this section, I will share my client's code and explain briefly.



```
data = pd.read_csv('df_1.csv', dtype=object)

where_are_NaNs = pd.isna(data)
data[where_are_NaNs] = 0
training_data, testing_data = train_test_split(data, test_size=0.2, random_state=25)

print(f"No. of training examples: {training_data.shape[0]}")
print(f"No. of testing examples: {testing_data.shape[0]}")

Y_train = training_data["Efficiency"]
X_train = training_data.drop(labels = ["egoid","Efficiency","timetobed","timeoutofbed", "bedtimedur", "minstofallasleep", "minsafterwakeup", "minsasleep",
 
Y_test = testing_data["Efficiency"]
X_test = testing_data.drop(labels = ["egoid","Efficiency","timetobed","timeoutofbed", "bedtimedur", "minstofallasleep", "minsafterwakeup", "minsasleep", "m

cols = X_train.columns

X_test = np.asarray(X_test).astype(np.float32)
Y_test = np.asarray(Y_test).astype(np.float32)
X_train = np.asarray(X_train).astype(np.float32)
Y_train = np.asarray(Y_train).astype(np.float32)
```

Figure 7.2 Data Preprocessing

Figure 7.2 describes preprocessing of the data. Firstly, I load my csv data to a dataframe and drop some columns that are used to calculate sleep efficiency. Then, I am splitting the data into train and test parts.



```
model = Sequential()
model.add(Conv1D(filters=32, kernel_size=8, activation='relu', input_shape=(82,1)))
model.add(MaxPooling1D(
    pool_size=2,
    strides=None,
    padding='valid',
    data_format='channels_last'
))

model.add(Conv1D(filters=32, kernel_size=8, activation='relu', input_shape=(82,1)))
model.add(MaxPooling1D(
    pool_size=2,
    strides=None,
    padding='valid',
    data_format='channels_last'
))

model.add(Conv1D(filters=32, kernel_size=8, activation='relu', input_shape=(82,1)))
model.add(MaxPooling1D(
    pool_size=2,
    strides=None,
    padding='valid',
    data_format='channels_last'
))

model.add(Flatten())

model.compile(optimizer=Adam(learning_rate=0.5), loss='mean_squared_error', metrics=['mse'])
```

Figure 7.3 CNN Model That Will be Applied to the Federated Learning

Then, I define my CNN model, this model is changing from architecture to architecture.

```
# Define Flower client

class FlowerClient(fl.client.NumPyClient):

   def get_parameters(self,config):

       return model.get_weights()



   def fit(self, parameters, config):

       model.set_weights(parameters)

       r = model.fit(X_train, Y_train, epochs=20, validation_data=(X_test, Y_test), verbose=0)

       hist = r.history

       print("Fit history : " ,hist)
```

```
        return model.get_weights(), len(X_train), {}


    def evaluate(self, parameters, config):

        model.set_weights(parameters)

        loss, mse = model.evaluate(X_test, Y_test, verbose=0)

        print("Eval accuracy : ", mse)

        return loss, len(X_test), {"accuracy": mse}


# Start Flower client

fl.client.start_numpy_client(

        server_address="localhost:"+str(sys.argv[1]),

        client=FlowerClient(),

        grpc_max_message_length = 1024*1024*1024

)
```

Figure 7.4 a Client for Federated Learning

Figure 7.4 defines a Flower client and starts it. The client is set to communicate with a Flower server that's expected to be running on the same machine (localhost) at the port provided through the command line argument sys.argv[1]. Here are the details:

The Flower client is implemented by extending the fl.client.NumPyClient class and implementing its three required methods: get_parameters, fit, and evaluate.

1. get_parameters(self, config) - This method is called by the Flower framework when the server requests the current model parameters from the client. This method returns the current model's weights as a NumPy array.
2. fit(self, parameters, config) - This method is called by the Flower framework when the server sends the updated global model parameters to the client for a new round of training. The method updates the local model's weights with these parameters and then fits the model to the client's local data. After the training, it returns the updated weights, the number of data samples, and an empty dictionary.
3. evaluate(self, parameters, config) - This method is called by the Flower framework

when the server requests an evaluation of the global model on the client's local data. The method sets the local model's weights to the parameters provided by the server and then evaluates the model. It returns the loss, the number of samples, and the accuracy.

The fl.client.start_numpy_client() function is then called to start the Flower client with the provided server address and client instance.

The grpc_max_message_length is set to 1GB, which is the maximum size of the messages that the client can send.

7.3 Execution

In order to run this FL architecture, we need to first run the server

- python3 server.py <port_number>

and then we need to open different terminals for each client and run these commands for each terminal.

- python3 client<client_number> <port_number>

Then the FL process will start.

8. Results

As mentioned above, We first run our CNN models centrally and then run in a federated manner. For federated learning, We created 2 experiments, one with 8 clients and the other with 4 clients. Table 1 shows traditional running results while Table 2 and Table 3 are showing the federated results for 8 clients and 4 clients, respectively. After presenting the results, We have prepared bar charts, to see the effectiveness of federated learning models compared to the traditional one.

Table 1 presents the result of conventional deep learning techniques

Table 1: Results of the Centralized Architectures

Architecture	MSE
C-P-FC-FC	0.0067
C-P-C-P-FC	0.0046
C-C-P-C-C-P-FC-FC	0.0041
C-P-C-P-FC-FC	0.0178
C-P-C-P-C-P	0.882

Table 2 presents the result of federated learning 8 clients and Table 3 presents the result of federated learning with 4 clients.

Table 2: Results of 8 clients Federated Architectures

Architecture	C1	C2	C3	C4	C5	C6	C7	C8	AVG
C-P-FC-FC	0.0045	0.012	0.0033	0.0034	0.0033	0.002	0.0021	0.0034	0.0042
C-P-C-P-FC	0.0054	0.014	0.0041	0.0041	0.0042	0.0019	0.0027	0.0042	0.005
C-C-P-C-C-P-FC-FC	0.0045	0.012	0.0043	0.0034	0.0035	0.0014	0.0022	0.0034	0.0043
C-P-C-P-FC-FC	0.0044	0.013	0.0036	0.0033	0.0033	0.0027	0.0021	0.0047	0.0046
C-P-C-P-C-P	0.088	0.086	0.089	0.089	0.088	0.089	0.089	0.088	0.088

Table 3: Results of 4 Clients Federated Architectures

Architecture	C1	C2	C3	C4	AVG
C-P-FC-FC	0.0091	0.0041	0.0032	0.0028	0.0048
C-P-C-P-FC	0.0099	0.0041	0.0032	0.0037	0.0052
C-C-P-C-C-P-FC-FC	0.009	0.0044	0.0025	0.0032	0.0047
C-P-C-P-FC-FC	0.0089	0.0037	0.0024	0.0029	0.0044
C-P-C-P-C-P	0.087	0.089	0.088	0.088	0.088

8.1 C-P-FC–FC

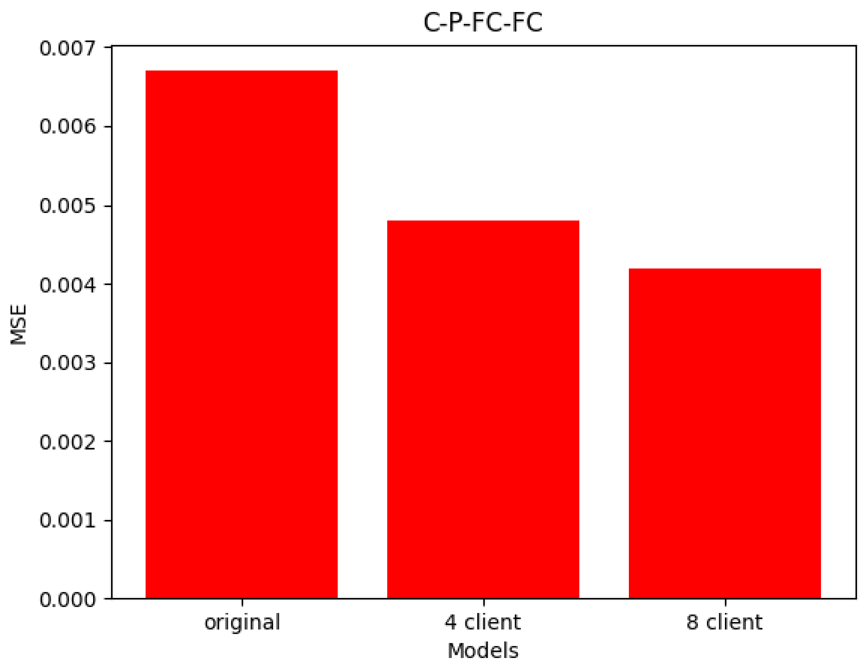


Figure 8.1 Performance Comparison for C-P-FC-FC Architecture

As shown in Figure 8.1, 4 client FL performed better than the original one and 8 client FL performed better than the 4 client FL However, the results are very close. Therefore, we can conclude that all the models performed similarly.

8.2 C-P-C-P-FC

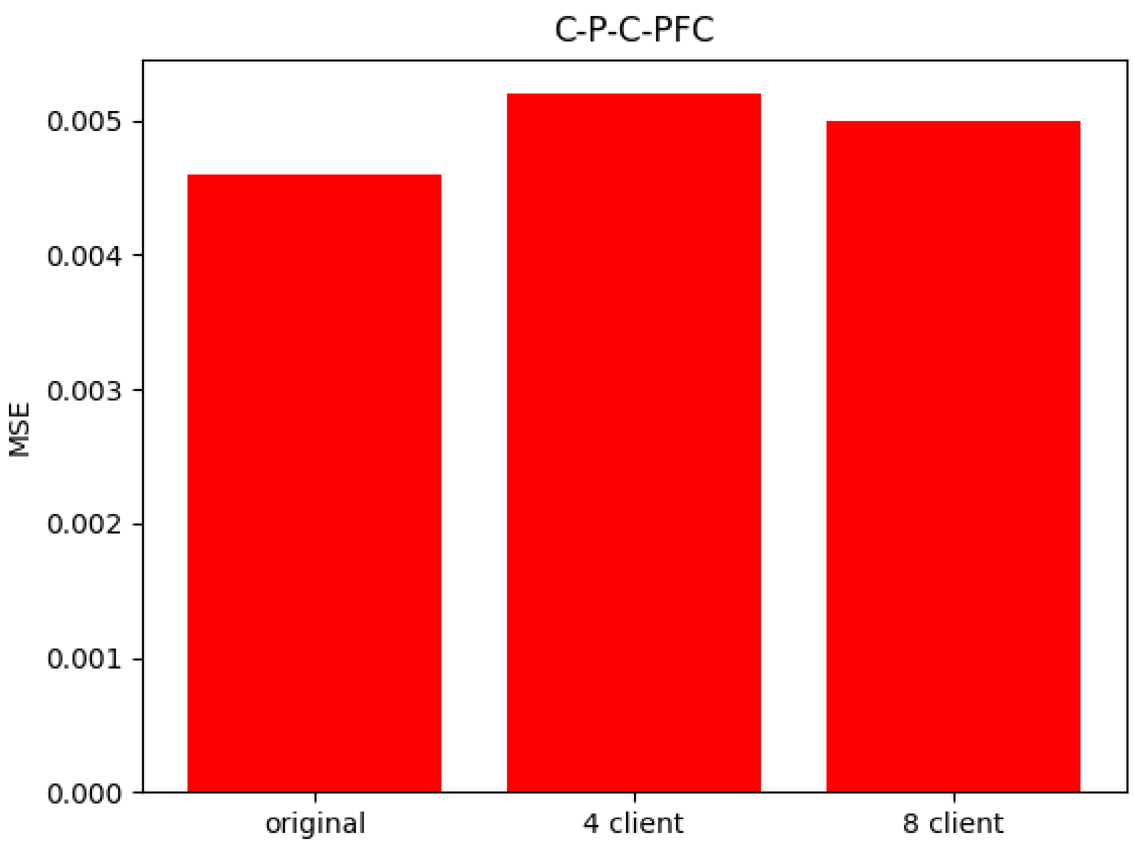


Figure 8.2 Performance Comparison for C-P-C-P-FC Architecture

As shown in Figure 8.2, the original model slightly performs better than both 4 clients model and 8 clients model. However, since MSE values are very close, we can say that the performance is not sufficiently different.

8.3 C-C-P-C-C-P-FC-FC

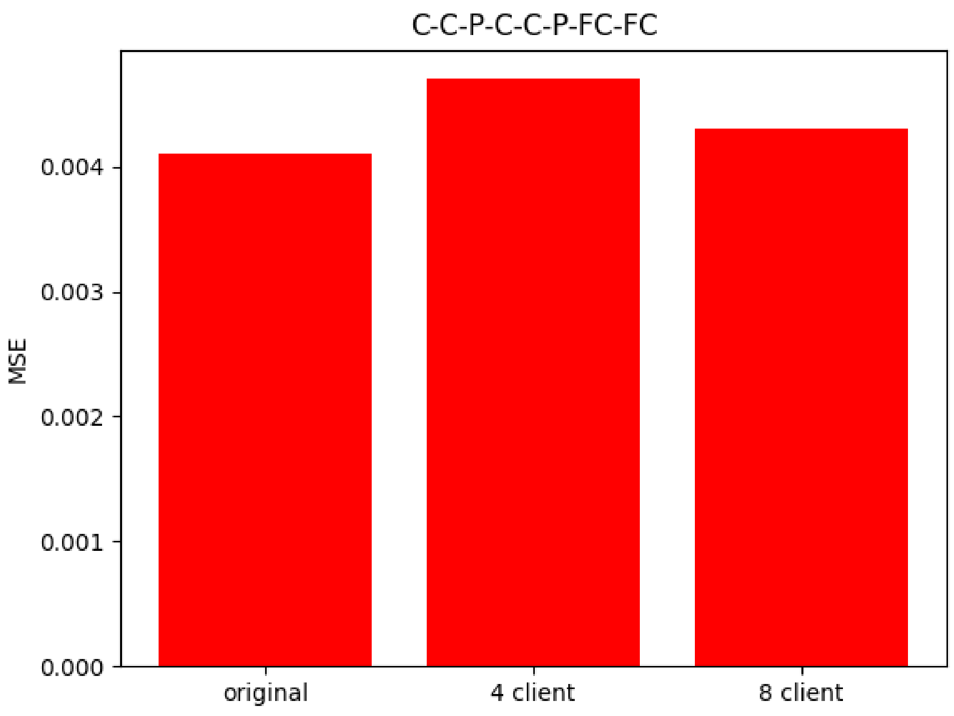


Figure 8.3 Performance Comparison for C-C-P-C-C-P-FC-FC Architecture

As shown in Figure 8.3, the original model slightly performs better than both 4 clients model and 8 clients model. However, since MSE values are very close, we can say that the performance is not sufficiently different.

8.4 C-P-C-P-FC-FC

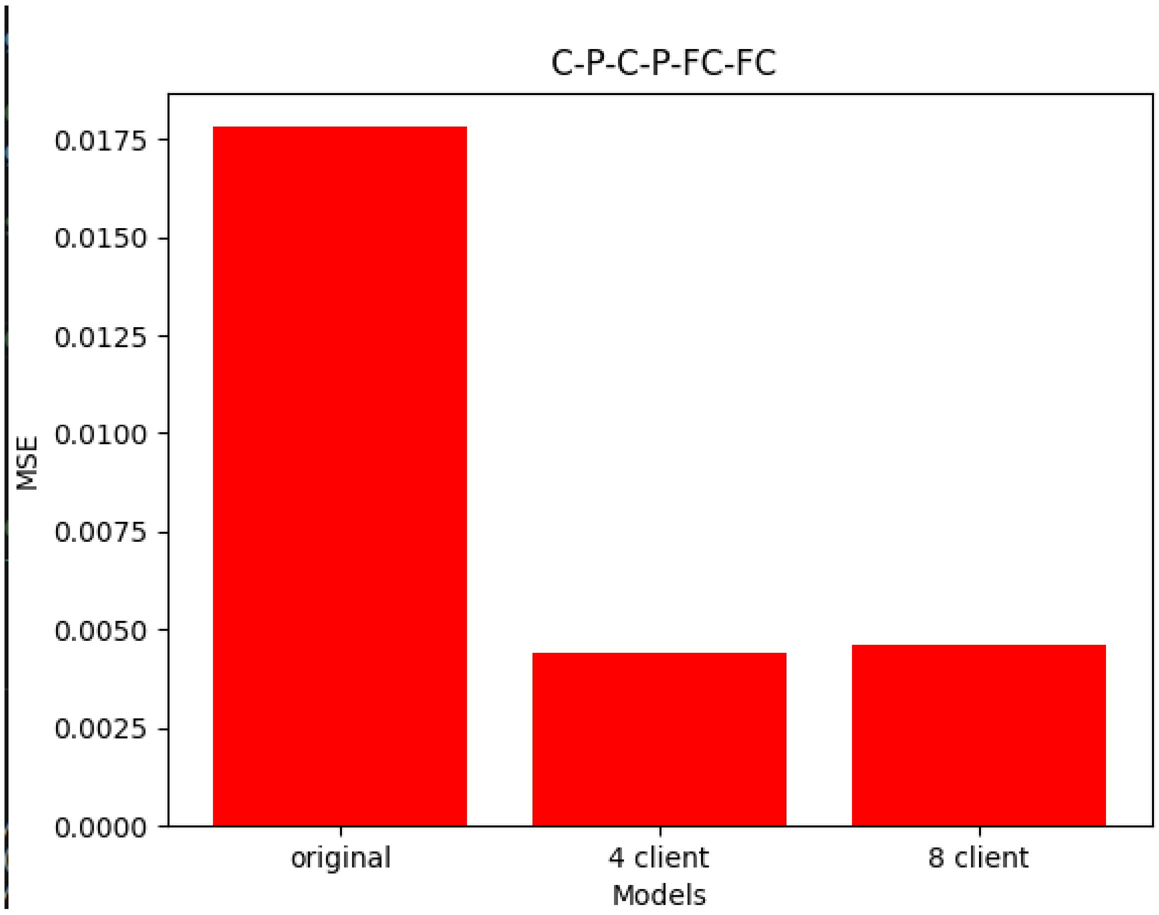


Figure 8.4 Performance Comparison for C-P-FC-FC Architecture

As shown in Figure 8.4, both federated models perform better than the original model. As seen on the chart, the difference is sufficient to deduce information about model comparison. When the original model performs poor, like MSE > 0.01, the federated models perform much better than the original one.

8.5 C-P-C-P-C-P

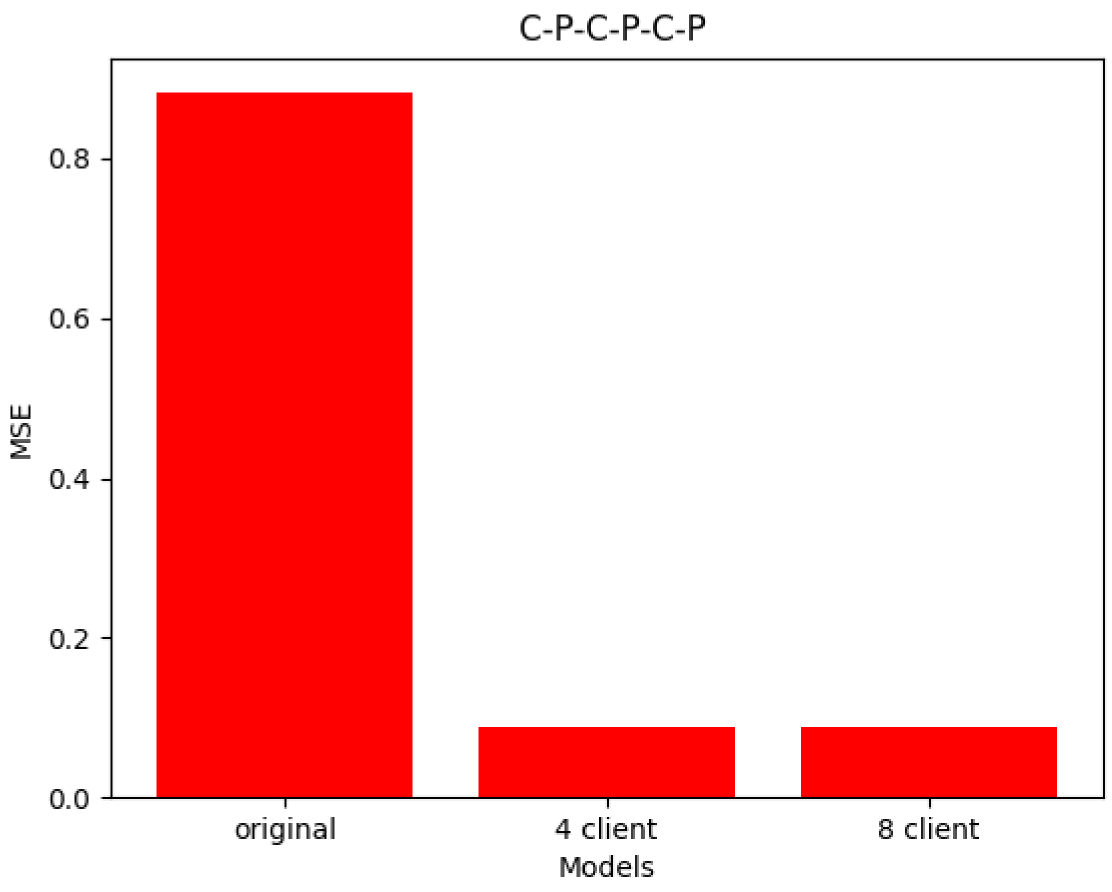


Figure 8.1 Performance Comparison for C-P-C-P-C-P Architecture

As shown in Figure 8.5, the MSE value of the original model is like 0.8 while federated ones have MSE value approximately 0.08. This indicates that federated models performed 10 times better than the original model. As we see in previous models, when the original model performs badly, like MSE value is higher than 0.01, then, federated models increase the performance dramatically.

9. Conclusion

In this project, we focused on applying federated learning techniques on CNN-based models to predict sleep quality and comparing the performances of centralized CNN-based architectures and federated CNN-based architectures.

We have described the results that we have obtained. For the first, second, and, third architectures the original model performs similar to the federated models, both 4 and 8 clients. However, for fourth and fifth models, federated models, both 4 and 8 clients, perform significantly better than the original model. One reason for this could be that the original model has significantly worse performance compared to first, second and third models and we may conclude that when the original model performs bad, federated learning models for this architecture can increase the performance. For the C-P-C-P-FC-FC model MSE value is greater than 0.01 unlike previous models, and federated models increased performance significantly. Similarly, for the fifth model, MSE value is greater than 0.8 and federated model increased performance 10 times. Thus, we can conclude that if the performance of a model is bad, federated learning can be applied and performance can be improved.

For the comparison of 4 clients and 8 clients federated models, 8 clients models have performed better than the 4 clients models except fourth architecture. However, the difference is very small as seen in the images, therefore, we can not say that the 8 clients model outperformed the 4 clients model significantly.

References

[1] Ordóñez, F.J.; Roggen, D. Deep convolutional and LSTM recurrent neural networks for multimodal wearable activity recognition. Sensors **2016**, 16, 115.

[2] Kılıç, O., Saylam, B., & İncel, Ö. D. (2023). Sleep Quality Prediction from Wearables using Convolution Neural Networks and Ensemble Learning. *arXiv preprint arXiv:2303.06028*.

[3] Beutel, D. J., Topal, T., Mathur, A., Qiu, X., Fernandez-Marques, J., Gao, Y., Sani, L., Li, K. H., Parcollet, T., de Gusmão, P. P. B., Lane, N. D. (2020). Flower: A Friendly Federated Learning Research Framework. arXiv preprint arXiv:2007.14390

[4] Konečný, J., McMahan, H. B., Yu, F. X., Richtárik, P., Suresh, A. T., Bacon, D. (2016). Federated Learning: Strategies for Improving Communication Efficiency. arXiv preprint arXiv:1610.05492