DeAI: A mobile application for privacy preserving decentralised machine learning in medical image classification.

Bachelor Project Report

Student: *Marcel Torné Villasevil*

Supervisors:

Professor Martin Jaggi

Dr. Mary-Anne Hartley

Abstract

Background

Standard machine learning techniques need centralized training data to build models. However, in many settings, such as medical applications, it is not feasible to build centralized datasets due to privacy concerns. This restriction limits the opportunities for collaboration in building machine learning models. Moreover, it puts actors with limited access to data and computing resources at a disadvantage. Federated machine learning has been proposed as a solution to this problem. It enables actors to collaborate and build complex machine learning models without sharing their sensitive data. However, in federated machine learning, actors rely on a central coordinator to aggregate the individual models and distribute the aggregated model back to the participants. Decentralized machine learning goes a step further and enables fully decentralized training without the need for a central coordinator. However, to the best of our knowledge, there is no widely used open-source privacy-preserving mobile application for decentralized machine learning that uses the current state-of-the-art approaches in the field.

Aim

To fill this gap, we propose a mobile-browser application, ***"DeAI",*** which allows users to collaboratively train models without sharing their data.

Methods

The platform was created using the *Vue.js* framework and incorporated a deep learning image classifier for the diagnosis of COVID-19 from Lung Ultrasound (LUS) images. To this end, the *DeepChest* model previously developed by our group was re-built using *TensorFlow.js*, simplified and tested on a simulated LUS dataset split across two and four users with various data distributions

Results

We present DeAI, a fully functional mobile-browser application with an intuitive user-friendly interface. Our simplified version of *DeepChest* achieved comparable results to the original model (AUROC 88.9% for the original vs AUROC 88.5% for the diagnosis of COVID on a given test set) ran in a browser. This performance was mostly maintained in a range of experimental non-independently with low bias on one of the labels and identically distributed data splits but performed poorly in heterogeneous settings where the bias for one label was of 90%. In this last case, our model overfits and always predicts the same label.

Conclusion

This work shows the potential of DeAI to act as a collaborative learning platform for robust medical image classification across distributed datasets without compromising user privacy.

INDEX

Background

The COVID-19 crisis has been further proof of the inequality among different countries around the world. This inequality also exists in the machine learning world. In low-resource settings, researchers do not have access to large datasets which are fundamental for developing complex models to solve major problems.

The problem

The problem that this work attempts to address is the fragmentation of datasets, whereby different entities might have different datasets for solving a common problem. However, they might not be able to share these datasets due to ethical restrictions and well-considered concerns about the intellectual property and privacy of data, especially in medical settings. Nevertheless, it would be beneficial for these different entities to combine their datasets to train more powerful models.

Existing solution to this problem

Nowadays, there exist several solutions for solving this problem. The first is to have a centralized setting, where each entity would share their data with a central server that will train a common model with all the data from the entities. The clear issue with this approach is that peers must trust a central node with their sensitive data. As mentioned before, in many cases such as in medical settings, this is not possible.

In recent years, there has been a new branch of machine learning algorithms called distributed learning. The most frequent type of distributed learning is called federated learning. The different entities (called nodes or peers) will each conserve the data exclusively on their side, and a central server coordinates the learning between them. On each iteration of the learning process, each node will share with the server the new weights of their training model. Then, the server will aggregate the different weights from each node and will share them back with the rest of the nodes. Finally, the nodes will proceed with their training using the newly received weights and continue iterating with the server.

As we can see, this is already a big improvement since now the central server will not have access to the private data of each node. However, we are still trusting a central node while training which is not desirable since it holds most of the power and will be the main beneficiary of this training process.

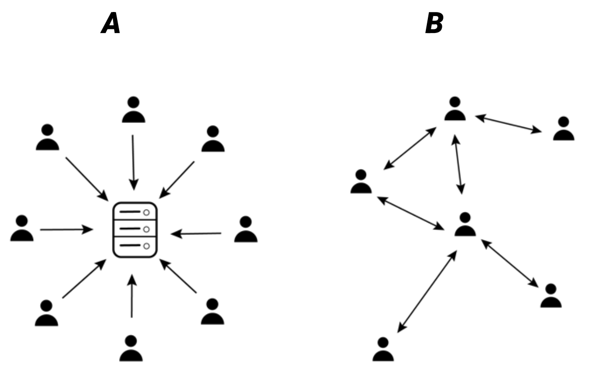


Figure 1. ***Architectures of (A) Federated (FL) and (B) Decentralised (DeL) learning.*** *Here, FL has a centralized server whereas DeL has no server.*

Therefore, there is a need for a mobile application for privacy-preserving, distributed machine learning that easily adapts to different types of tasks. Thus, we built a mobile browser application, called DeAI, that implements a decentralized learning protocol. Contrasting with federated learning, decentralized learning lacks a central coordinator. The different nodes communicate directly in a peer-to-peer fashion, on an underlying communication network topology, while attempting to perform gradient-based learning[1]. As we can see this form of communication is convenient since we do not need to trust a central server.

AIM and objectives

We have three main objectives for this project.

1. To build a mobile browser application to train deep learning models in a decentralized network.
2. To create an intuitive user interface to ensure access to users without prior knowledge of decentralized learning.
3. To apply this application to a real-world use case of a medical image classification task and test its performance in various conditions of data distribution.

Objective 1: Decentralized Learning

Build a modularizable code base on which decentralized learning can take place using an interchangeable permutation of newly integrated models, and modifiable communication patterns and aggregation methods.

Objective 2: Intuitive user interface

Firstly, the interface should be accessible in a browser from all devices, from laptops to mobile phones, it should be clean (i.e. minimal, so as to reduce perceived complexity), but at the same time provide sufficient information that engineers and researchers will need to train their models. In addition, it is necessary to provide constant visual feedback to our users so that they can be aware of the flow of the app. Finally, the interface should provide separate functions for image and tabular dataset tasks.

Objective 3: Real world use case experimentation

To demonstrate the utility of *DeAI* we will apply it to the real-world case of COVID-19 diagnosis from an image dataset of Lung Ultrasounds (LUS): the LUS-COVID dataset from *iGH*[2] which comprises LUS images from 162 patients suspected of COVID-19. Each patient has a set of images from different parts of the lung and is labelled as COVID-19 positive or COVID-19 negative.

[1] *Source: Martin Jaggi, 2020, Algorithms for Decentralized Artificial Intelligence*
[2] *Source : https://github.com/epfl-iglobalhealth/LUS-COVID-main/tree/master/dataset , iGH, EPFL.*

Methods

The app was developed, benchmarked, and tested using two well-known datasets: 1) the titanic dataset for tabular tasks and 2) the MNIST dataset for image classification tasks. Once the functionality was verified on these tasks, the LUS-COVID dataset was implemented.

Application interface and backend

Figure 2 shows the final mock-up of the app showcasing its main functionalities and screens. Previous versions of the mock-up were shared with prospective users to receive feedback and further improve the design. After several iterations we arrived at the before mentioned final version in Figure 2.

The platform was built using *Vue.js*, a *JavaScript* framework for building dynamic single page applications on the browser. A single-page application means that there's only a single HTML fetched by the browser, but at the same time allowing different screens. The advantage of building a single page app is that it will allow users to train different models or distribute different model weights concurrently, since threads are maintained when switching between screens which would be much harder in a multiple-page app.

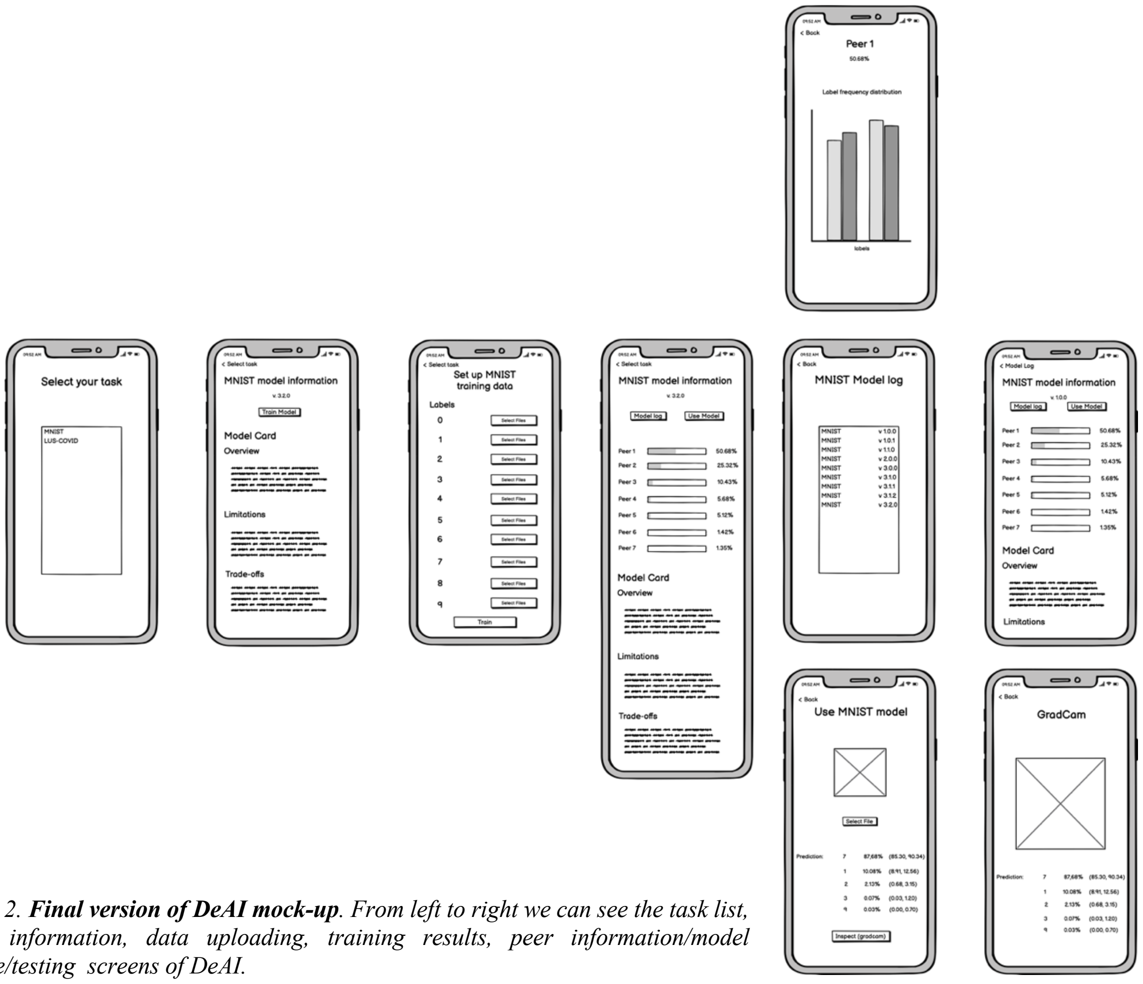


Figure 2. ***Final version of DeAI mock-up***. *From left to right we can see the task list, model information, data uploading, training results, peer information/model storage/testing screens of DeAI.*

Moreover, using *Vue.js* makes it easy to deploy the application using *GitHub Pages*, which makes it very convenient since we can use continuous integration and make the changes available to our users instantaneously.

To create an intuitive interface, we used *TailwindCSS*, a framework for making visually appealing browser applications through CSS. Among other features, it allows the user to deploy dark mode and to personalize the interface.

The main component of the app is a router (nomenclature in *Vue.js* for the component that organizes the links between screens in the app). This router will link the different components, of which we have four (ordered by the flow of the app):

- **Task list:** where all the available tasks are fetched from a server and displayed.
- **Model Description:** where some basic information about each model can be found. Here, the user is prompted to select the model they want to use. This last view will only be displayed if the user has a saved model.
- **Model training:** here, the user can upload the data to train the model and choose whether they want to train the model locally or in a distributed manner. While the model is being trained, the user will be able to see how the training and validation accuracies evolve throughout training. Moreover, if training distributively, there will be additional information displayed like the number of peers that the user is helping, and how much time the user is waiting to receive the weights. After training the model, the user will be able to save it and/or go to the testing component if desired.
- **Model testing**: here, the user will upload the testing dataset to test the newly trained model. If the user only uploads one data point, the results will be displayed on the screen. In the case where they upload more than one data point, a csv file containing the predictions will be downloaded.

Each task has different models and different pre-processing pipelines. For modularisation, we decided to have the components explained above generalized and reused for all tasks. These will be using different task scripts depending on which task has been selected. A task script consists of the following functions:

- **create_model**: creates a *Tensorflow.js* model to solve the task.
- **data_preprocessing**: handles the pre-processing of the data points and will return the pre-processed training set and the labels one-hot-encoded.
- **predict**: passes the data points to the model and return its predictions

Moreover, the task script also contains two objects with information about the task:

- **display_information**: which contains texts explaining the architecture and limitations of the model, an overview of the task, and a sample datapoint.
- **training_information**: which contains necessary details for training the model, like batch size, learning rate, optimizer, number of epochs and some attributes related to decentralised training like the server's port and the threshold of the number of peers to wait for.

Finally, an important part of the work has been in refactoring the app and adding documentation to make an app that will be easy to further improve and extend in the future.

Decentralised learning

There are three main components to the decentralised learning feature:

1. The communication protocol
2. The network topology
3. The weights aggregation method

The communication protocol uses *PeerJS* and a small extension library developed previously by the *DeAI* team[3] to communicate the model weights between peers and communicate with the server. *TensorFlow* models are represented via two files, one containing the model's weights and the other containing the model architecture. We use the team's *PeerJS* extension library to serialize them, send them to other peers and convert them back into *TensorFlow* models once received.

Regarding the network topology, at the end of a training epoch, each peer sends a weight request to the rest of the peers, sends his/her model to those who requested it and waits to receive as many weights as specified by the threshold parameter. If some weights do not arrive in ten seconds, the peer continues training without them.

```
Algorithm 1: Communication between peers
 Result: $net$
 Initialise connection to PeerJS server;
 connect as peer $p_j$;
 Get list of Peers connected $L$;
 Initialize $net$ with random weight matrix $W_i^0$;
 for $t \leftarrow 0$ to $epochs$ do
     Sample points $\mathcal{E}$;
     $W_i^{t+\frac{1}{2}} \leftarrow UpdateStep(W_i^t, \mathcal{E})$;
     $RequestWeights(L)$;
     $W_{0..L}^t \leftarrow ReceiveWeights(L)$;
     $W_i^{t+1} \leftarrow W_i^{t+\frac{1}{2}}$;
     $totalWeights \leftarrow 0$;
     for $j = 0$ to $j = L$ do
         if $W_j^t \neq NULL$ then
             $W_i^{t+1} \leftarrow W_i^{t+1} + W_j^t$;
             $totalWeights \leftarrow totalWeights + 1$
         end
     end
     $W_i^{t+1} \leftarrow W_i^{t+1} / totalWeights$
 end
```

Figure 3. ***Communication algorithm****. The communication algorithm consists in getting the list of connected peers for the desired task and start training the model collaboratively. When training, we are doing a simple average among the received weights at the end of each epoch.*

[3] *PeerJS DeAI - MLO:* https://github.com/epfml/DeAI/tree/master/experiments/peerjs

For model aggregation, we compute an average between all the received weights and the current weight. This has proven to be an effective weight aggregation method[4] and is simple to implement.

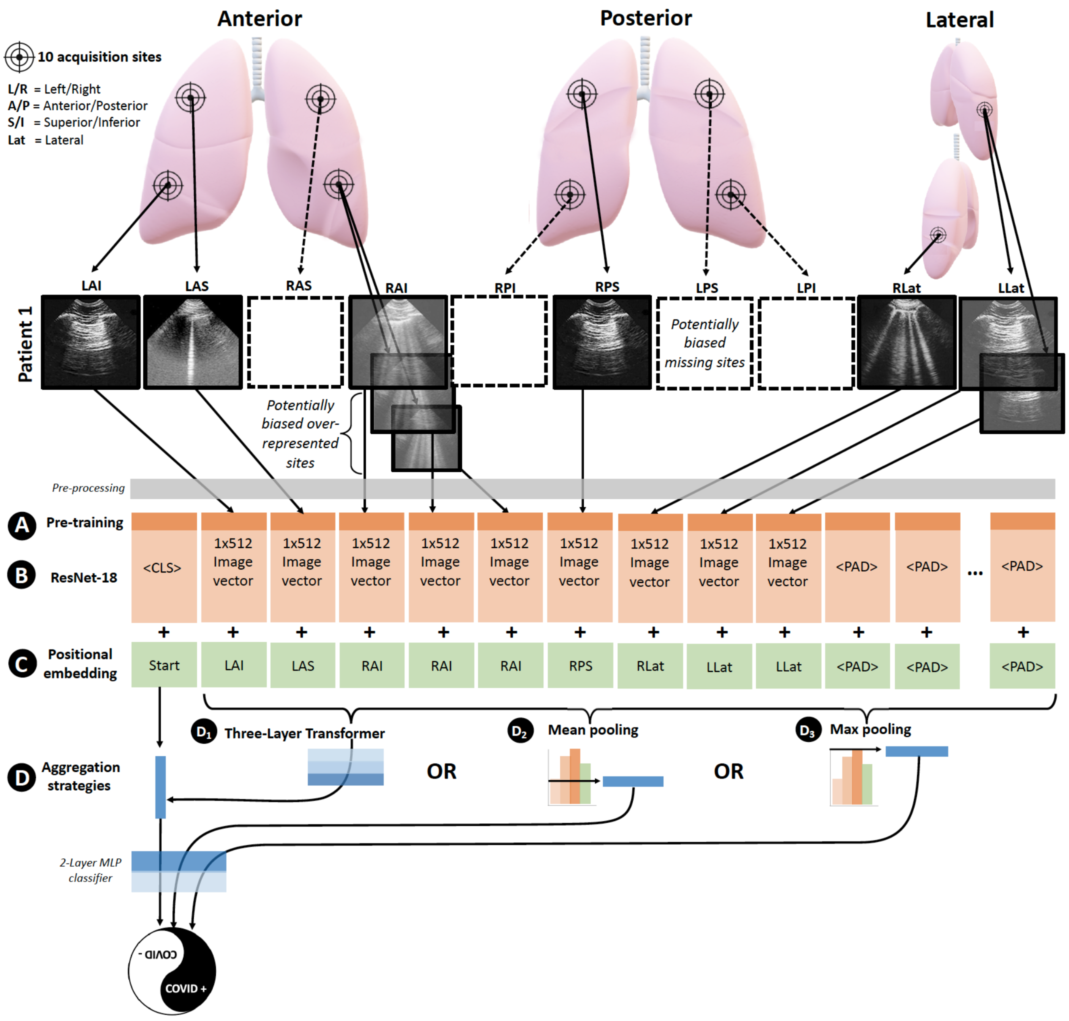
The final algorithm for the communication between peers is shown on figure 3. There are some small adaptations due to this algorithm having to be run on a real-world case. For example, even if *L* peers have been connected to the server, there could be some problems on their side and the peer might not receive the weights for a long time. To solve this problem, we set a maximum waiting time of ten seconds, after which the peer does not wait for the weights anymore and continues to the next step. Moreover, we also set a threshold value so that if for example a peer only wants to train with two more peers, they can set the threshold to two and then as soon as it gets two weights the algorithm will proceed into the next step and hence it will not have to wait for the rest of the peers.

LUS-COVID model

We adapt an existing deep learning classifier (DeepChest) to detect patterns of COVID-19 in the LUS dataset. To create the model and do the pre-processing, we use *TensorFlow.js*, which is the well-known python library *TensorFlow* adapted to be run on the browser and coded in *JavaScript*. Running models on the browser is relatively new, which means that not many optimizations are in place at the moment and most of the time training complex python models that run on GPUs is not possible in a mobile phone's browser. For this reason and due to time constraints, we implemented a simplified version of *DeepChest*. The original *DeepChest* uses a *ResNet18* to perform feature extraction of the images, but this model is not available in *Tensorflow.js* due to its size. For this reason, we used *Mobilenet* to perform the feature extraction. This is an optimized architecture made of convolutional neural networks and pooling layers trained on *ImageNe*t but much smaller and executable on a phone's browser. Due to time limitations, we did not implement the positional embeddings in our model. Moreover, we only implemented one of the aggregation functions available in *DeepChes*t (i.e. mean pooling)

The final *DeAI's DeepChest* model architecture is presented in figure 4. It consists of using *Mobilenet* as a feature extractor for each image, followed by a mean pooling layer that will average all images from the given patient. Finally, we pass the resulting vector into a two-layer multi-layer perceptron (*MLP*) classifier.

[4] Nedić, A. et al. (2018)



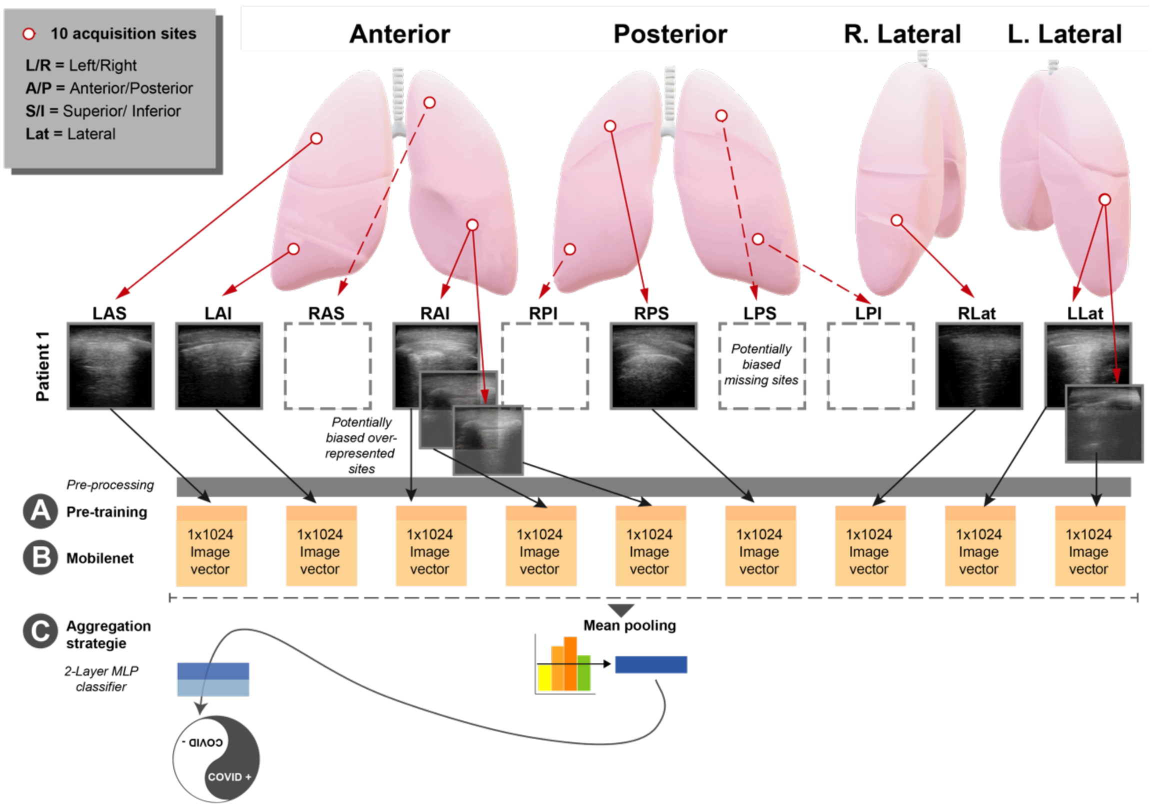


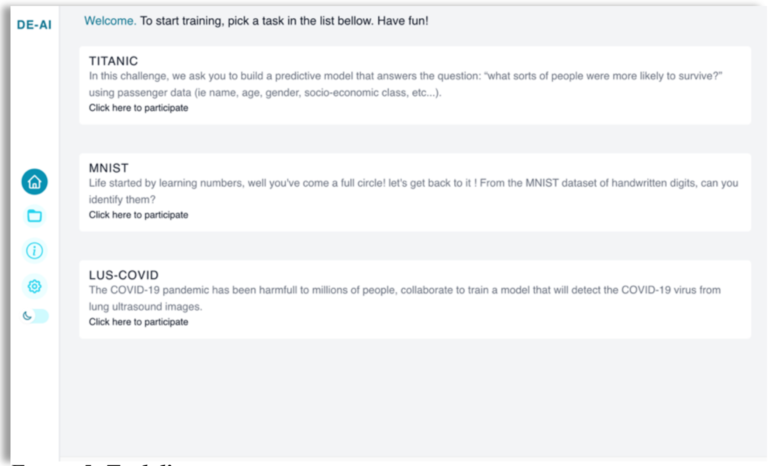
Figure 4. ***Original (top) vs DeAI's (bottom) DeepChest architectures.*** *Adapted from iGH.*

Results

Application interface

Task list screen

Figure 5 shows *DeAI's* main screen. All tasks that are currently available for training are displayed here. The user can click on the desired task to proceed to the overview screen.



DE-AI

Welcome. To start training, pick a task in the list bellow. Have fun!

TITANIC
In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc...).
Click here to participate

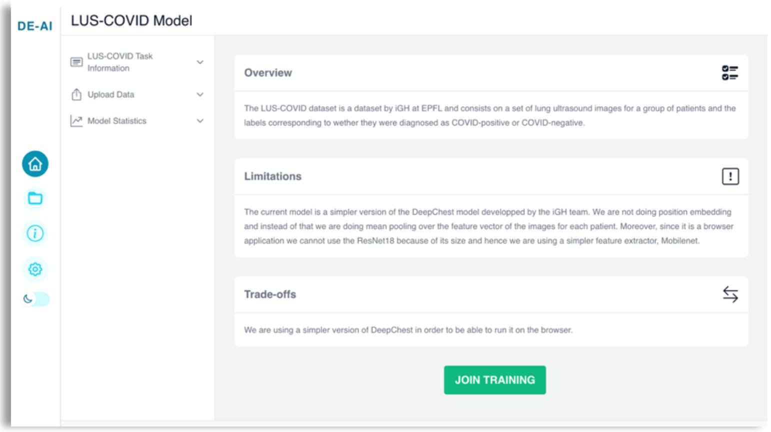
MNIST
Life started by learning numbers, well you've come a full circle! let's get back to it ! From the MNIST dataset of handwritten digits, can you identify them?
Click here to participate

LUS-COVID
The COVID-19 pandemic has been harmfull to millions of people, collaborate to train a model that will detect the COVID-19 virus from lung ultrasound images.
Click here to participate

Figure 5. ***Task list screen***

Overview screen

Figure 6 shows the overview screen for a specific task. It gives some useful information about the task and the model designed for this task.



DE-AI

LUS-COVID Model

Overview

The LUS-COVID dataset is a dataset by iGH at EPFL and consists on a set of lung ultrasound images for a group of patients and the labels corresponding to wether they were diagnosed as COVID-positive or COVID-negative.

Limitations

The current model is a simpler version of the DeepChest model developped by the iGH team. We are not doing position embedding and instead of that we are doing mean pooling over the feature vector of the images for each patient. Moreover, since it is a browser application we cannot use the ResNet18 because of its size and hence we are using a simpler feature extractor, Mobilenet.

Trade-offs

We are using a simpler version of DeepChest in order to be able to run it on the browser.

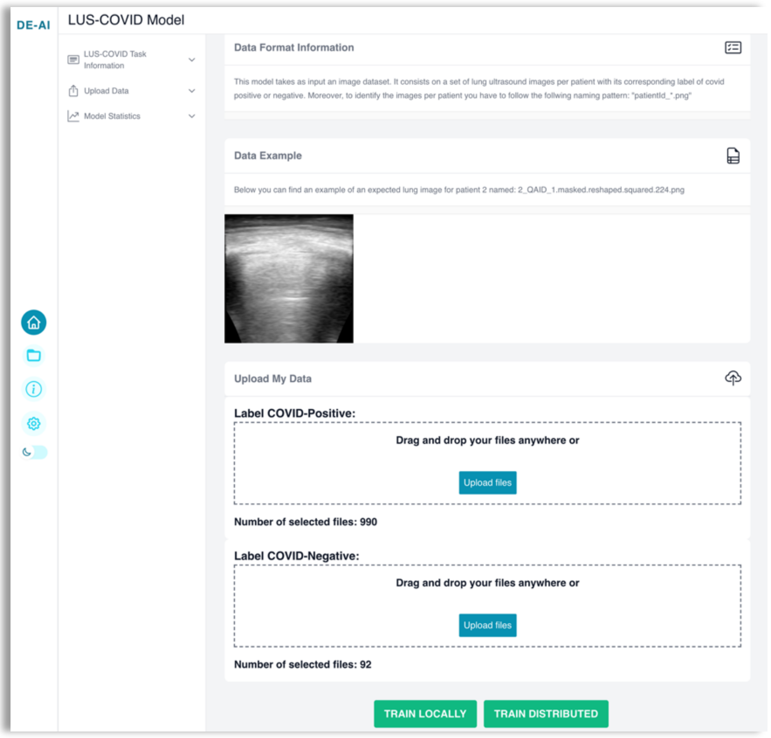
Figure 6. ***Overview screen***

Training images screen

In Figure 7 we see the training screen, where the users can upload their dataset to train the model. In the case of an image dataset, the users will upload the images in the appropriate input box according to the label for each image (one input box per label). In the case where the task is a tabular dataset task, there will be just one input box to upload the csv file

(Note: there are some extra features in the tabular dataset tasks such as changing the name of the columns to match the expected ones by the task's script).

Once the data is uploaded, the user will be able to select either the "Train



DE-AI

LUS-COVID Model

Data Format Information

This model takes as input an image dataset. It consists on a set of lung ultrasound images per patient with its corresponding label of covid positive or negative. Moreover, to identify the images per patient you have to follow the follwing naming pattern: "patientId_*.png"

Data Example

Below you can find an example of an expected lung image for patient 2 named: 2_QAID_1.masked.reshaped.squared.224.png

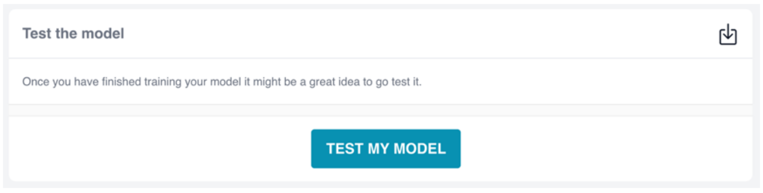
Upload My Data

Label COVID-Positive:
Drag and drop your files anywhere or
Upload files
Number of selected files: 990

Label COVID-Negative:
Drag and drop your files anywhere or
Upload files
Number of selected files: 92

Figure 7. ***Training screen***

Locally" button to train the model individually or the "Train Distributed" button to proceed with distributed learning. When finished training, the users can proceed to test their model by clicking on the "Test my Model" button at the bottom of the screen (Figure 8).

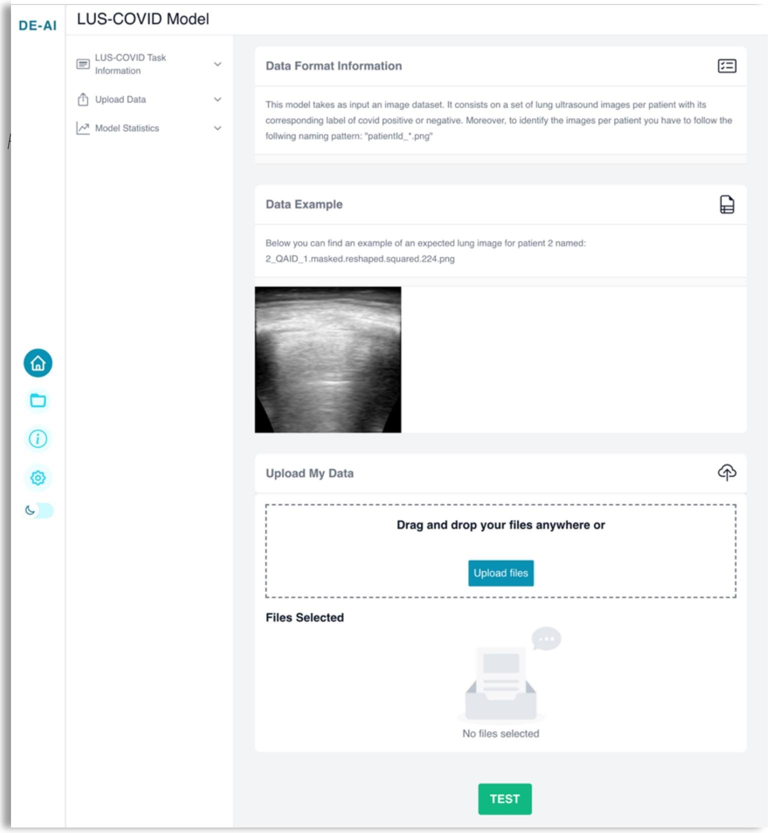

Test the model

Once you have finished training your model it might be a great idea to go test it.

Figure 8. ***Test my model button***

Testing images screen

In this screen (Figure 9), the user will be able to upload his/her testing set to test the newly trained model. Once the testing set is uploaded into the upload box, the user can click on the "Test" button and receive the predictions. In the case where more than one data point is provided, the predictions will be downloaded in a predictions.csv file. Otherwise, the prediction will be displayed below this button.


LUS-COVID Model

Data Format Information

This model takes as input an image dataset. It consists on a set of lung ultrasound images per patient with its corresponding label of covid positive or negative. Moreover, to identify the images per patient you have to follow the follwing naming pattern: "patientId_*.png"

Data Example

Below you can find an example of an expected lung image for patient 2 named: 2_QAID_1.masked.reshaped.squared.224.png

Upload My Data

Drag and drop your files anywhere or

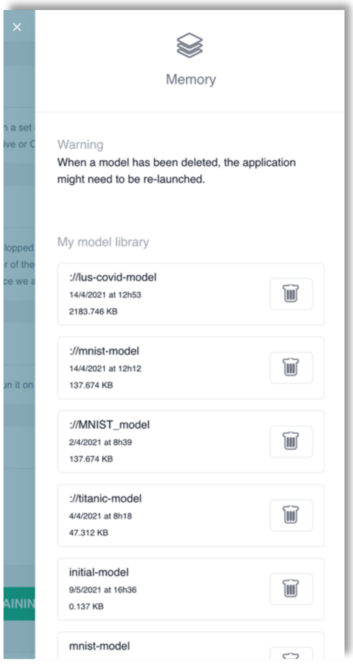
Files Selected

No files selected

Figure 9. ***Testing screen***

Model storage

As mentioned earlier we provide the capability of storing models. At this point, the model storage system is simple, there are at most 2 models per task at any given point in time: the working model and the saved model. The working model is the newly created or newly trained model and the saved model is a model that the user stored in some previous session. The users can see and remove the models they saved, by clicking on the folder item in the left bar (Figure 10) and the user can save the model after training (Figure 11). Moreover, users have the possibility to choose which of the two models they want to use in the task overview screen (Figure 12).


Memory

Warning
When a model has been deleted, the application might need to be re-launched.

My model library

://lus-covid-model
14/4/2021 at 12h53
2183.746 KB

://mnist-model
14/4/2021 at 12h12
137.674 KB

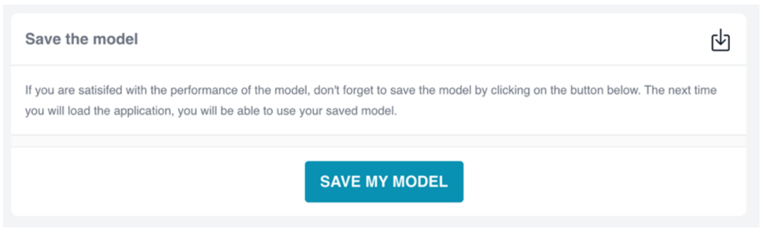
://MNIST_model
2/4/2021 at 8h39
137.674 KB

://titanic-model
4/4/2021 at 8h18
47.312 KB

initial-model
9/5/2021 at 16h36
0.137 KB

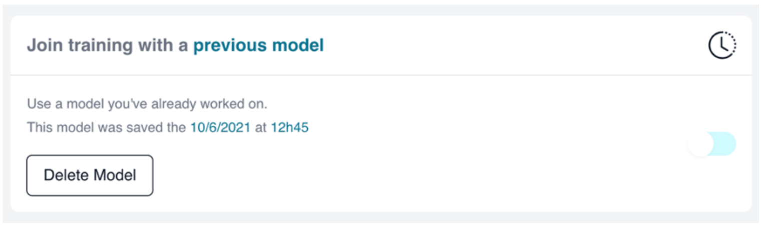
mnist-model

Figure 10. ***Model storage sidebar screen***


Save the model

If you are satisifed with the performance of the model, don't forget to save the model by clicking on the button below. The next time you will load the application, you will be able to use your saved model.

*Figure 11. **Save model button**. Button located at the bottom of the training screen*

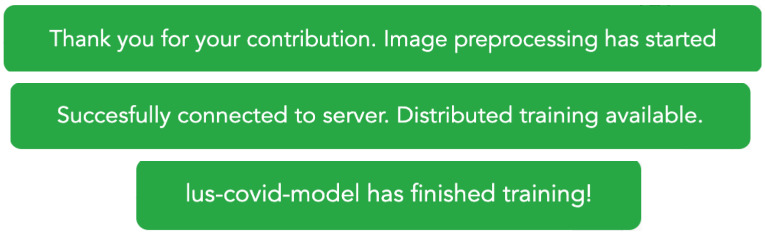

Join training with a previous model

Use a model you've already worked on.
This model was saved the 10/6/2021 at 12h45

*Figure 12. **Select previous model button.** Button located at the bottom of the task overview screen*

Feedback to the user

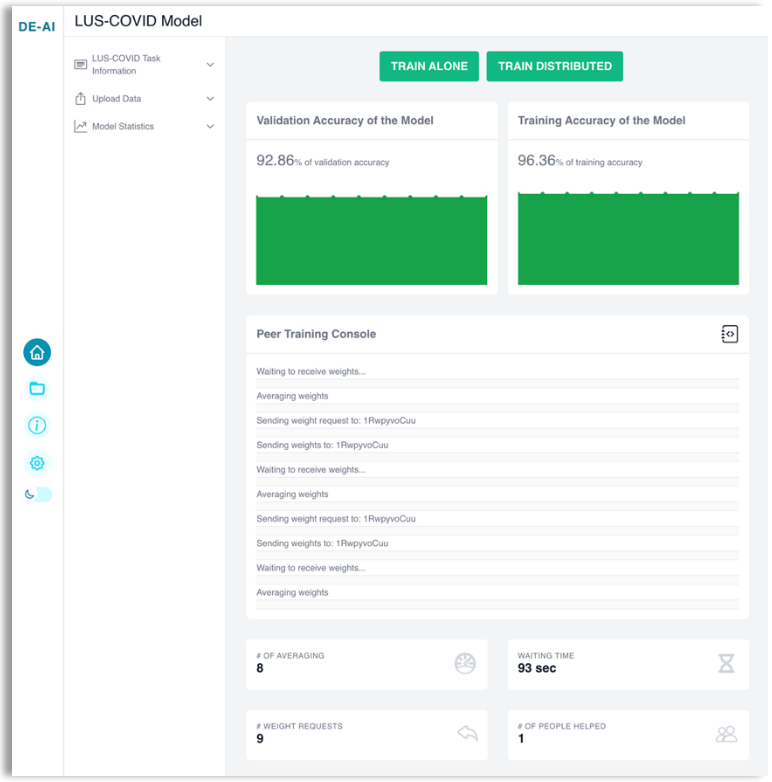
One of the main goals of this project is to have an application that is understandable to users with various levels of expertise (being both simple to understand and providing enough information for more experienced users so that they may comprehend their training process and improve quality of their dataset, if necessary, e.g. by pre-processing the images before training). To satisfy this goal, provide as much feedback as possible. We do so in two different ways, the first is by displaying small messages at the bottom right of the screen informing the user about the state of the application (Figure 13). The second is by updating dynamically the UI with new information. We mainly do this in the training screen. As shown in Figure 14, we provide information about the training and validation accuracy of the model at each epoch for both locally and distributed learning.


Thank you for your contribution. Image preprocessing has started

Succesfully connected to server. Distributed training available.

lus-covid-model has finished training!

*Figure 13. **Feedback state messages to the user***



Validation Accuracy of the Model
92.86% of validation accuracy

Training Accuracy of the Model
96.36% of training accuracy

Peer Training Console

Waiting to receive weights...
Averaging weights
Sending weight request to: 1RwpyvoCuu
Sending weights to: 1RwpyvoCuu
Waiting to receive weights...
Averaging weights
Sending weight request to: 1RwpyvoCuu
Sending weights to: 1RwpyvoCuu
Waiting to receive weights...
Averaging weights

OF AVERAGING
8

WAITING TIME
93 sec

WEIGHT REQUESTS
9

OF PEOPLE HELPED
1

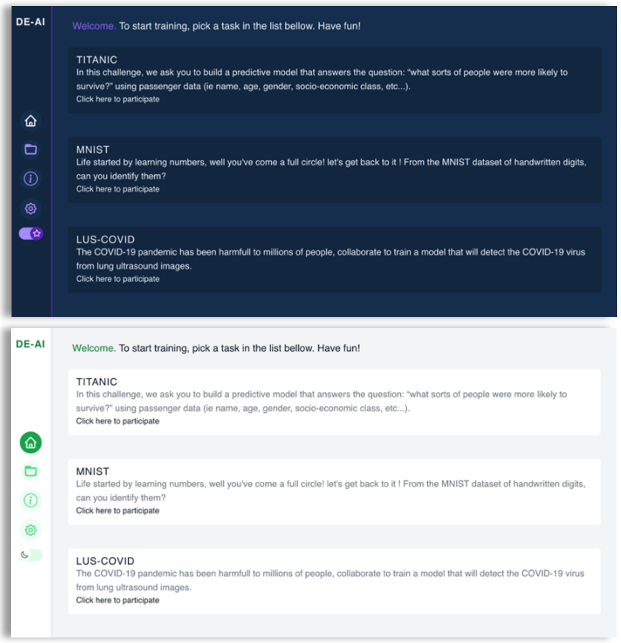
*Figure 14. **App training screen***

Furthermore, when training distributively we provide information such as the number of peers helped, waiting time as well as a training console where information about the weights shared and received is displayed.

Customization

To make the interface as appealing as possible to all users, we provide the possibility to customize it. There are two levels of customization. First, the user can choose to have the background colour in either dark or white mode. In addition, the user can also set the theme colour to one of those shown in Figure 15. All this is available by clicking on the settings button.

You can click into the link below to play with the live version of the DeAI.[5]


DE-AI

Welcome. To start training, pick a task in the list bellow. Have fun!

TITANIC
In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc...).
Click here to participate

MNIST
Life started by learning numbers, well you've come a full circle! let's get back to it ! From the MNIST dataset of handwritten digits, can you identify them?
Click here to participate

LUS-COVID
The COVID-19 pandemic has been harmfull to millions of people, collaborate to train a model that will detect the COVID-19 virus from lung ultrasound images.
Click here to participate

DE-AI

Welcome. To start training, pick a task in the list bellow. Have fun!

TITANIC
In this challenge, we ask you to build a predictive model that answers the question: "what sorts of people were more likely to survive?" using passenger data (ie name, age, gender, socio-economic class, etc...).
Click here to participate

MNIST
Life started by learning numbers, well you've come a full circle! let's get back to it ! From the MNIST dataset of handwritten digits, can you identify them?
Click here to participate

LUS-COVID
The COVID-19 pandemic has been harmfull to millions of people, collaborate to train a model that will detect the COVID-19 virus from lung ultrasound images.
Click here to participate

Figure 11. ***Screen personalization examples****. Same screen on dark and light mode.*

LUS-model

Aiming to verify the accuracy of *DeAI's DeepChest* model we split the dataset in 5 folds. Then we iterate in a round robin fashion to select one split as the testing set and the remaining splits are used as training sets. We obtained the five ROC curves in Figure 16 and averaged their areas to get the final accuracy. The presented results are achieved by a Python implementation of our simplified *DeepChest* model. The original *DeepChest* model using a transformer achieves an area under the *ROC* curve of 88.9%[6] while our model achieves the accuracy of 88.5%.

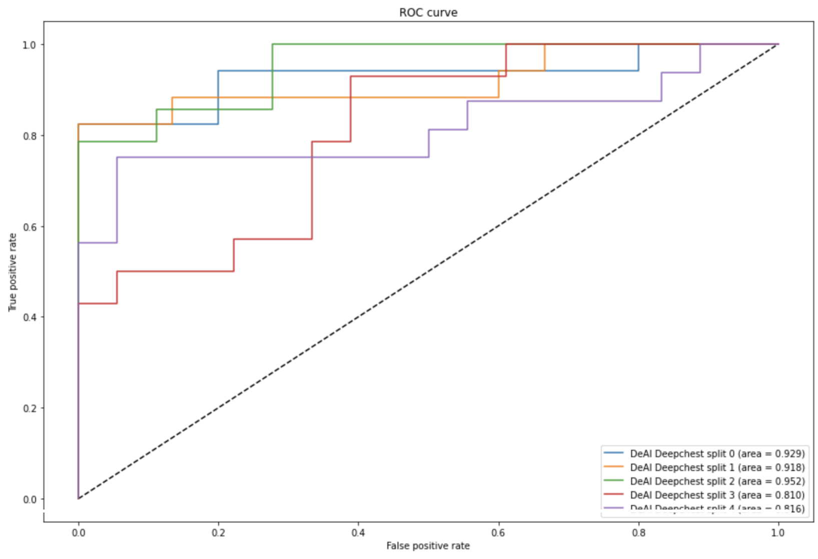


Figure 12. ***DeAI's DeepChest ROC curve.*** *Here, we see the five ROC curves for the DeAI's DeepChest version. These were obtained by splitting the dataset in five splits, using four splits for training, one for testing and swapping the training and testing splits in a round-robin fashion per each iteration*

These are surprisingly good results since our model is greatly simplified compared to the more complex *DeepChest* with transformer architecture, which is run with more resources, python's *TensorFlow*, and a *ResNet18* and the results are almost the same.

[5] *Link to DeAI's live application:* [*https://epfml.github.io/DeAI/*](https://epfml.github.io/DeAI/)
[6] [*https://github.com/epfl-iglobalhealth/LUS-COVID-main/blob/master/notebooks/validation_noValidation_evaluation.ipynb*](https://github.com/epfl-iglobalhealth/LUS-COVID-main/blob/master/notebooks/validation_noValidation_evaluation.ipynb)

Decentralised learning

To demonstrate that *DeAI* is working as anticipated, and that peers can benefit from using decentralised learning by building better models collaboratively, we ran some benchmarks on the app using the *LUS-COVID* dataset, and *DeAI's DeepChest* model. The first step was to extract 10% of the dataset into a training to be used as a testing set. The second step was to split the remaining of the dataset set among the different peers. We split the data in different manners, homogeneously and heterogeneously, to see how decentralized learning would perform in each situation. For each peer, the received dataset is split into training and validation set. Then, we proceeded by running each training set for each peer locally and verify its performance on the testing set. Afterwards, we train a new model collaboratively and again verify its performance on the testing set. Next, we will present the results for the four experiments, splitting the dataset uniformly distributed, quantity biased, fever biased and covid biased:

Uniformly distributed experiment

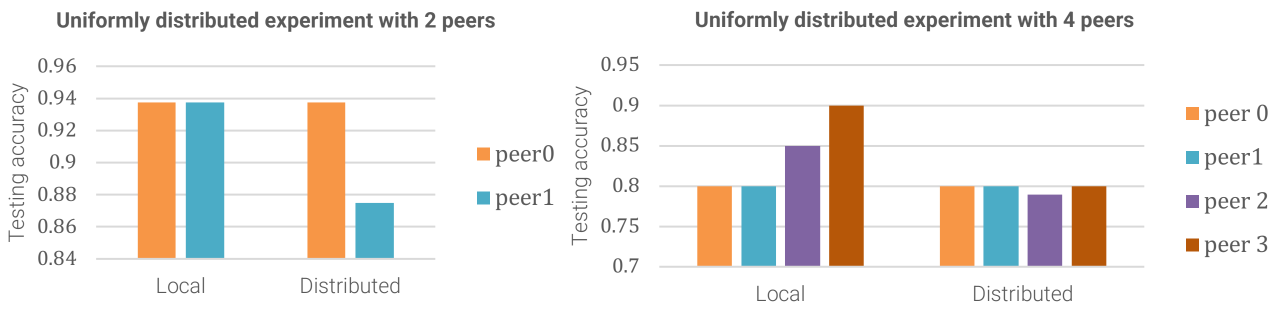


Figure 13. ***Bar plot of uniformly distributed experiment with 2 and 4 peers***

In this experiment, we split the dataset such that each peer has the same number of data points with a uniform distribution in the labels. When two peers are training, we see a slight decrease (6%) in performance for one of the peers, and similarly for when four peers are training. However, there is an interesting observation which is that when training four peers, we see that the model for each peer has the same performance, which is an indication that the model converged. Indeed, this is a good example of how fragmented datasets trained locally can have non-representative and spurious performances due to low sample numbers. Statistically, we expect each model trained locally on random splits to be equivalent. Distributed training achieves this outcome.

Quantity biased experiment

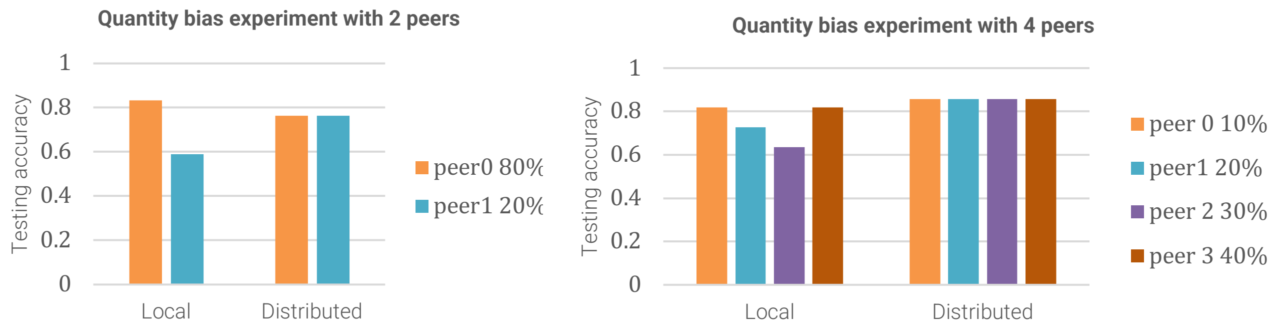


Figure 14. ***Bar plot of quantity biased experiment with 2 and 4 peers***

When the dataset is split unevenly in terms of sample number, the performance of the distributed option shows potential improvement compared to the local training. We observe the anticipated result that for two peers, the performance for the one with more data (80%) is better than the one with less data (20%). In the case of four peers the performance of all peers in a distributed setting is preferable to local training. Finally, we observe again the same accuracy in the final model for each peer which is an indicator of the convergence.

Fever-biased experiment

In this experiment we mimic a label skew by biasing the dataset according to whether the patients had fever or not (implying a slight bias in COVID +/- patients). Thus, peer 0 (with no fever) has more patients who were covid negative. Here, training distributively has a performance superior to local training and even improves in comparison to centralised training (93.3% accuracy vs 88.5% found for the ROCAUC area previously). We can deduce that they combine their knowledge and achieve a final model that is much better than the individual ones.

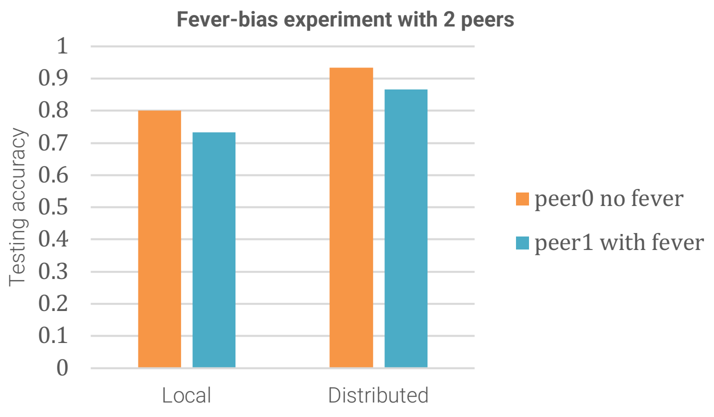


Figure 19. ***Bar plot of fever-biased experiment with 2 peers***

COVID-biased experiment

In this last experiment we perform a more direct label skew where peer 0 has 90% COVID+ and peer 1 10%. In Figures 21-22, we can see that while they are training, the second peers completely bias the first peer. And the first peer will perform badly, even in his/her own training and validation sets. Moreover, when we investigate the predictions,

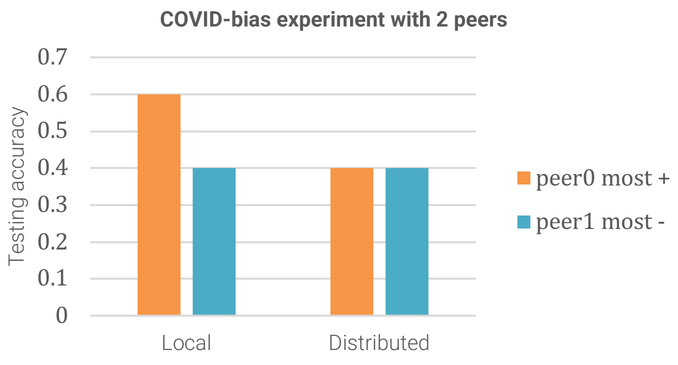


Figure 20. ***Bar plot COVID-biased experiment with 2 peers***

both peers predict the same label, covid negative, for all the testing patients, which reinforces the point made before.

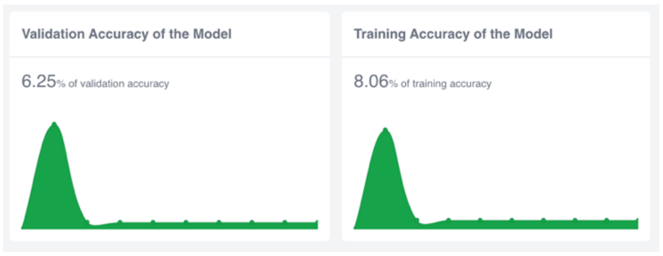


Figure 21. ***Peer 0 training curve for the COVID-biased experiment.***

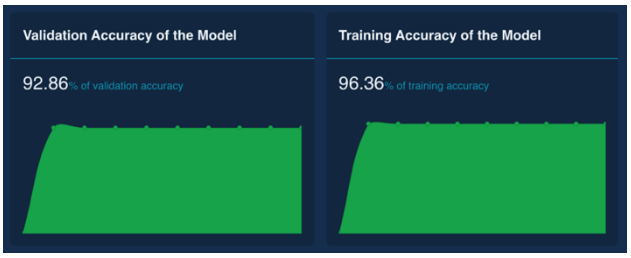


Figure 22. ***Peer 1 training curve for the COVID-biased experiment.***

Compute efficiency

Finally, we wanted to conclude with some numbers on the time overhead that comes from using decentralized learning. From the experiments we ran, we can say that usually the time overhead for waiting to receive weights and update them is around 5 seconds per epoch. However, if the weights lower limit threshold (which is how many weights you want to receive before the averaging) is too high then this overhead can reach the upper limit of ten seconds per epoch. However, we have observed that when training distributively the model seems to converge in less iterations than when training locally.

Discussion

Limitations

As has been mentioned before, the app has some limitations with respect to the sizes of the models being trained and the resources available since the models are trained on browsers and mobile phones. However, we have not reached the limit yet. We have observed that if we are careful with memory management, we can in fact train larger models on larger datasets. For example, during the data pre-processing pipeline, we must be extra careful compared to when we train models in python, by properly deleting tensors when not used anymore. Moreover, we believe that loading training data in batches will also provide great performance benefits.

In the decentralised learning part, we also found several limitations. The first is that for the moment there is no control on which state of the training a peer is currently in, for example, a peer might just be finishing the training which means that his/her model is already providing good performance, but it might start getting weights from a peer that just started training and for which the weights are almost random. This will decrease the performance of the first peer's model. The second limitation is that for the moment the weight exchange is solely done at the end of each epoch, however, for some cases it would be beneficial to exchange weights in the middle of the epochs, as for the covid-bias experiment.

In addition, during this project, we started the app from scratch, as explained before we even made the mock-ups, and the time we had was limited so there are plenty of new features that could bring even more value to the app that we were not able to implement.

Future work

We are excited and optimistic about *DeAI* since it is setting the foundations to having a great mobile application for privacy preserving decentralised machine learning for any kind of task. Just to give an overview at what will come next:

- *Personalized decentralised learning*: we will be able to implement Frédéric Berdoz's algorithm on the application and this will bring many new capabilities. It will improve incentivisation and will allow us to provide feedback to the peers about the quality of the data they are providing. We plan to display this information in the form of a peers ranking. Moreover, it will also allow the possibility of doing weighted averaging so that peers' models that improve your model in your validation set will have higher weights than the ones which perform poorly.
- *Enhance privacy*: one of the core goals of this app and of using decentralised learning is the privacy of the data. At MLO, Milos Vujasinovic is working on building an algorithm that will enhance the privacy of the data so that no information about each peer's dataset can be deduced from the model weights.
- *Tasks with bigger and more complex models*: Martin Milenkoski is already working on implementing Relay SGD in the app to be able to train bigger and more complex models for more complicated tasks as CIFAR10.

These are just three next steps that will be pursued in the near future, and which really excite us, but this is just the beginning of *DeAI* and there is much room for continuing to improve and extend it further.

Acknowledgments

The application described in this report was a joint effort by Paul Mansat (PM) and Marcel Torné Villasevil (MTV), the author of the report. PM worked on the tabular dataset tasks and on implementing the Titanic task. PM also was the main contributor to the storage model system, the communication protocol, and the aggregation method for models and the UI we currently have. MTV worked on the image datasets and on implementing MNIST, he also built and designed the *DeAI's DeepChest* version together with the LUS-COVID task and was the main contributor for the testing screen. Moreover, MTV implemented an improved version of the model aggregation, which is the currently deployed one, and made some fixes to the communication protocol. MTV is also the author of the Mock-ups, the experiments and benchmarks presented in this report. PM and MTV contributed together to building, modularising and refactoring the architecture of the app.

I would like to thank Paul for being an incredible teammate while developing this application throughout this semester, it has been a real pleasure working with him. I would also like to thank my supervisors, Professor Martin Jaggi and Dr. Mary-Anne Hartley, for all their useful advice and mentorship, it has been a real honour to be their supervisee. Moreover, I would also like to thank them for trusting me to work on such an interesting and exciting project. Finally, I would like to thank Martin Milenkoski who joined back *DeAI* in the middle of the semester and has really helped us to improve several aspects of the application, in improving the modularisation, and now by bringing new features to the application.

References

Jaggi, M. (2020) *DeAI – Algorithms for Decentralized Artificial Intelligence. Part B2: The scientific proposal.* ERC Consolidator Grant 2020

Nedić, A. et al. (2018) *"On Distributed Averaging Algorithms and Quantization Effects."* IEEE Transactions on Automatic Control 54 (2009): 2506-2517. Online: https://arxiv.org/pdf/0711.4179.pdf

Berdoz,F. (2021) *Quantifying Peers Contribution in Fully Decentralized Learning*, *Semester Project at iGH* – MLO, EPFL, unpublished work.

Vujasinovic, M. (2021*) HyperAggregate: A sublinear secure aggregation protoco*l, *Semester Project at DeAI* – MLO, EPFL, unpublished work.

Data source repository:

- *Deepchest repo:* https://github.com/epfl-iglobalhealth/LUS-COVID-main/tree/master/deepchest
- *DeAI repo:* https://github.com/epfml/DeAI
- *LUS-Covid dataset* https://github.com/epfl-iglobalhealth/LUS-COVID-main/tree/master/dataset
- *LUS-DeAI benchmarking & DeAI's Deepchest model repo:* https://github.com/epfl-iglobalhealth/LUS-DeAI
- *PeerJS DeAI - MLO:* https://github.com/epfml/DeAI/tree/master/experiments/peerjs

Marcel Torné Villasevil and Paul Mansat (2021) *DeAI live version*: https://epfml.github.io/DeAI/