AIM-HI report

Model chosen

```
def create_model():


    # CIFAR10 model
    model = models.Sequential()
    model.add(layers.Conv2D(32, (3, 3), activation='relu',
↪   input_shape=(32, 32, 3)))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.MaxPooling2D((2, 2)))
    model.add(layers.Conv2D(64, (3, 3), activation='relu'))
    model.add(layers.Flatten())
    model.add(layers.Dense(64, activation='relu'))
    model.add(layers.Dense(10))

    model.compile(optimizer='adam',

↪   loss=tf.keras.losses.SparseCategoricalCrossentropy(from_logits=True),
              metrics=['accuracy'],
              #run_eagerly=True #debug only
              )

    return model
```

The model chosen is the same as the FL model. Everything used default settings when necessary. Optimizer is `adam` and loss is calculated using Sparse Categorical Crossentropy.

Voting and Consensus function

The voting and consensus function used for the experiment is unanimous voting. That is unless all participants agree on the label, the image won't be part of the training set for the current iteration.

```
def new_vote(voters, images, labels, test_image, test_labels):

    #print(images[0])

    f = open(filename, "a")

    global_predictions = []
    for i, v in enumerate(voters):
        global_predictions.append(v.predict(images))
        #check_learner_acc(v, images, labels)
        results = v.evaluate(test_image, test_labels,
↪   batch_size=128, verbose=0)
        print(f"results of voter {i} acc test: loss={results[0]}
         ↪  acc={results[1]}")
        #print(len(images), len(labels))
        print(f"{e},{i},{results[0]},{results[1]}", file = f)

    global_predictions = np.array(global_predictions)
    f.close()

    #print(global_predictions)

    # Voting part loop
    image_voted     =       []
    label_voted     =       []
    image_not       =       []
    label_not       =       []

    certain_global = []
    count = 0

    for i in range(len(labels)):
        tmp = np.zeros(10)

        for cg in global_predictions:
            best = np.argmax(cg[i])
            tmp[best] += 1 #select only the best and check that
↪   its equal to 5 ie unanimous vote

        if tmp[np.argmax(tmp)] == len(voters):
```

```
            image_voted.append(images[i])
            label_voted.append(labels[i])

            if np.argmax(tmp) == labels[i]:
                count += 1
        else:
            image_not.append(images[i])
            label_not.append(labels[i])

#    check_vote(global_predictions, certain_global, label_voted)

    return [image_voted, label_voted], [image_not, label_not],
     ↪  count
```

A file is created and each time this function is called, the loss and accuracy of each model is recorded on that file with the following format:

```
Epoch,Learner,Loss,Accuracy
0,0,1.7411577365875244,0.3758000135421753

---SNIP---

1,4,1.7231868793487548,0.3840000033378601
2,0,1.4522437152862548,0.49869999289512634
```

Model training loop

```
# Training loop iterations
while len(global_x) != 0 and e<epochs:
    print(f"Training epoch {e}")

    e += 1

    voted, remaining, count = new_vote(learners,
                                       global_x,
                                       global_y,
                                       x_test, y_test)

    global_x = np.array(remaining[0])
    global_y = np.array(remaining[1])

    # fit model to the new labels
    # Training loop
    for j in range(len(learners)):
        print(f"Training learner {j}")

        tmp_img = trainsets[j][0]
```

```
        tmp_labels = trainsets[j][1]

        #print(tmp_labels)
        #print(certain_global)

        trainsets[j][0] = np.append(tmp_img,
                                    voted[0],
                                    axis=0)
        trainsets[j][1] = np.append(tmp_labels, voted[1], axis=0)

        #assert len(trainsets[j][0]) == len(trainsets[j][1])

        train_local(trainsets[j][0], trainsets[j][1], [], j, e)
↪   #old code
        #train_local(voted[0], voted[1], learners, j)
        learners[j] =
↪   load_model(f'models/new_method/model_{j}.tf')
```

The server has all the unlabeled data and sends all of it to the participants. Using unanimous voting, each particiapnt only trains on the images that were labeled and the rest is kept for the next iteration. We continued this loop until there was no data left to be trained on or after a certain number of epochs, which ever came first.

Train local function

```
def train_local(train_x, train_y, learners, i, epoch_num):

    model = create_model()

    model.fit(train_x, train_y, epochs=10, shuffle=True,
↪   verbose=0)

    # Maybe better way but needed to save into a file at one
    ↪  point
    model.save(f'models/new_method/model_{i}.tf')
    if i == 0:
        model.save(f'models/epochs/model_{epoch_num}_{i}.tf')
```

One problem with TF that happened was that a pre-trained model, the accuracy on the test set would start going down. The fix to this semmed to create an entirely new model and train it again. Each model trains on the dataset it owns and the one that was classified for 10 epochs before the next communication round.

Settings

The settings can be modified before the voting loop under the `Global Vars` header.

```
##############
# Global Vars
##############

"""
0 = all messages are logged (default behavior)
1 = INFO messages are not printed
2 = INFO and WARNING messages are not printed
3 = INFO, WARNING, and ERROR messages are not printed
"""
#os.environ['TF_CPP_MIN_LOG_LEVEL'] = '2'

# Use GPUs
tf.config.set_soft_device_placement(True)
#tf.config.run_functions_eagerly(True) # debug for model not
↪ compiling
#tf.debugging.set_log_device_placement(True) #uncomment if need
↪ to check that it is executing off of GPU
tf.get_logger().setLevel('ERROR')

filename = "outputs/plotdata_new_algo_1K_local.csv"

f = open(filename, "a")
f.write("Epoch,Learner,Loss,Accuracy\n")
f.close()

(x_train, y_train), (x_test, y_test)=
↪ keras.datasets.cifar10.load_data()

x_train = x_train/255.0
x_test = x_test/255.0

train_size = 40000

assert train_size < len(x_train)

trainsets, global_x, global_y, local_ds  =
↪ dataset_formatting(x_train, y_train, train_size, 10, 5)
#trainsets, global_x, global_y =
↪ dataset_formatting_label_culling(x_train, y_train, 20000,
↪ True, 0.0)
```

```
# Set number of itterations either via local_ds or number of
↪  epochs to train
epochs = 30
```

`train_size` is the size of the unlabeled datasets that each participant will have to label and then train on.

Summary:

```
- 5 clients
- 40K unlabeled
- optimizer = adam (default settings)
- batch_size = 128
- commrounds = 600
- comm period =  evaluation period = 20
```

Results obtained:

The results obtained for the first model (`model_0.tf`) are compiled in the google spreadsheet under the AIM-HI Tab:

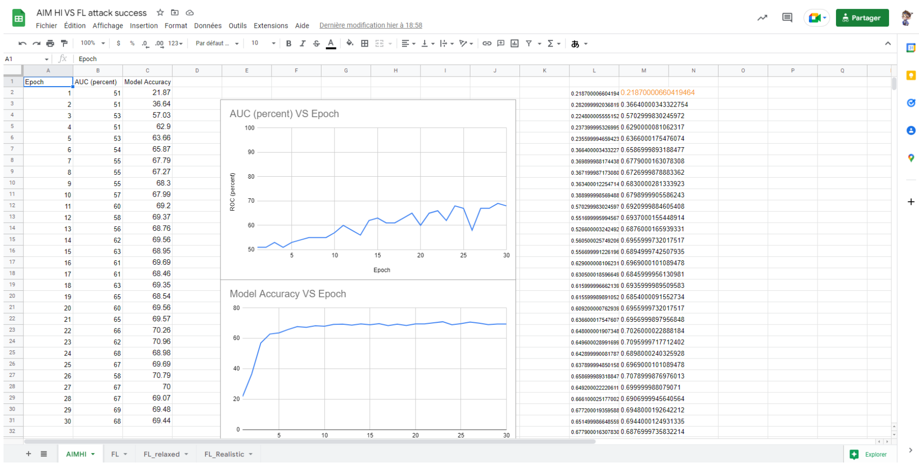

Figure 1: google sheets

The full file containing the loss and accuracy of each models in under the `outputs` folder and the name is `plotdata_new_algo_1K_local.csv` (only model_0's results are in the table)

FL report

Bash file

```
#! /bin/bash --

method="FL"
numclients=5
numrounds=600
commperiod=10
evaluationrounds=20
trainbatchsize=500
clients="PaperCIFARNet"
dataset="CIFAR10"

python3 -u experiment.py --method $method --client $clients
↪ --dataset $dataset --num-clients $numclients --num-rounds
↪ $numrounds --train-batch-size $trainbatchsize --comm-period
↪ $commperiod --evaluation-rounds $evaluationrounds | tee
↪ Exp${method}_${client}_${dataset}_${numclients}cl_n${numrounds}_r${numrounds}.log
```

No other modifications of the code were made and this was run to create all the models for the FL experiment.

PyTorch to TF conversion

```
import torch
import torch.nn as nn
import torch.nn.functional as F
import torch.optim as optim
import torch.nn.init as init
from torchvision import datasets, transforms
from torch.autograd import Variable

import onnx
from onnx_tf.backend import prepare

import numpy as np
from IPython.display import display
from PIL import Image

import tensorflow as tf

from pytorch2keras.converter import pytorch_to_keras

import os
import re
```

```
class Cifar10PaperNet(nn.Module):
    def __init__(self):
        super(Cifar10PaperNet, self).__init__()
        self.conv1 = nn.Conv2d(3, 32, kernel_size=3)
        self.pool = nn.MaxPool2d(2, 2)
        self.conv2 = nn.Conv2d(32, 64, kernel_size=3)
        self.conv3 = nn.Conv2d(64, 64, kernel_size=3)
        self.fc1 = nn.Linear(1024, 64)
        self.fc2 = nn.Linear(64, 10)

    def forward(self, x):
        x = self.pool(F.relu(self.conv1(x)))
        x = self.pool(F.relu(self.conv2(x)))
        x = F.relu(self.conv3(x))
        x = torch.flatten(x, 1) # flatten all dimensions except
↪   batch
        x = F.relu(self.fc1(x))
        x = self.fc2(x)
        return x


regex = r"model_round(.*).model"

directory = 'torch_models'

for filename in os.listdir(directory):
    f = os.path.join(directory, filename)
    #print(filename)

    if os.path.isfile(f):

        epoch = int(re.search(regex, filename).group(1))
        epoch = (epoch+1)//20
        #print(epoch)

        trained_model = Cifar10PaperNet()
        trained_model.load_state_dict(torch.load(f))

        input_np = np.random.uniform(0, 1, (1, 3, 32, 32))
        input_var = Variable(torch.FloatTensor(input_np))
        print(input_var.shape)

        tf_ref = pytorch_to_keras(trained_model, input_var, [(3,
↪   32, 32,)], verbose=False, change_ordering=True)
        print(tf_ref.summary())
```

```
tf_ref.save(f'keras_models/model_{epoch}.tf')
```

Since the PyTorch model couldn't be tested for the privacy, we had to convert them using `onnx` and `pytorch2keras`. This script allowed us to covnert these models to a format that would be readable for the ml privacy tool.

TF model	PyTorch converted model

TF model:

```
Model: "sequential_5"
_________________________________________________________________
Layer (type)                 Output Shape              Param #
=================================================================
conv2d_15 (Conv2D)           (None, 30, 30, 32)        896
_________________________________________________________________
max_pooling2d_10 (MaxPooling (None, 15, 15, 32)        0
_________________________________________________________________
conv2d_16 (Conv2D)           (None, 13, 13, 64)        18496
_________________________________________________________________
max_pooling2d_11 (MaxPooling (None, 6, 6, 64)          0
_________________________________________________________________
conv2d_17 (Conv2D)           (None, 4, 4, 64)          36928
_________________________________________________________________
flatten_5 (Flatten)          (None, 1024)              0
_________________________________________________________________
dense_10 (Dense)             (None, 64)                65600
_________________________________________________________________
dense_11 (Dense)             (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
_________________________________________________________________
```

PyTorch converted model:

```
Layer (type)                 Output Shape              Param #
=================================================================
input_0 (InputLayer)         [(None, 32, 32, 3)]       0
_________________________________________________________________
11 (Conv2D)                  (None, 30, 30, 32)        896
_________________________________________________________________
12 (Activation)              (None, 30, 30, 32)        0
_________________________________________________________________
13 (MaxPooling2D)            (None, 15, 15, 32)        0
_________________________________________________________________
14 (Conv2D)                  (None, 13, 13, 64)        18496
_________________________________________________________________
15 (Activation)              (None, 13, 13, 64)        0
_________________________________________________________________
16 (MaxPooling2D)            (None, 6, 6, 64)          0
_________________________________________________________________
17 (Conv2D)                  (None, 4, 4, 64)          36928
_________________________________________________________________
18 (Activation)              (None, 4, 4, 64)          0
_________________________________________________________________
19_CHW (Lambda)              (None, 64, 4, 4)          0
_________________________________________________________________
19 (Flatten)                 (None, 1024)              0
_________________________________________________________________
20 (Dense)                   (None, 64)                65600
_________________________________________________________________
21 (Activation)              (None, 64)                0
_________________________________________________________________
output_0 (Dense)             (None, 10)                650
=================================================================
Total params: 122,570
Trainable params: 122,570
Non-trainable params: 0
```

NB: the _CHW layer that is added on the converted PyTorch model is because PyTorch and Tensorflow read image data differently. This layer is to account for that difference.

Results

Similar to the AIM HI results, all of the results pulled from these models are under the FL tabs of the same spreadsheet.