CMSC 828A Homework 2: Ensemble Learning, Federated Learning

Josh Davis and **Benjamin Wu** and **Shelvin Pauly** and **Michel Temgoua**
UMD
Computer Science Department
8125 Paint Branch Dr, College Park, MD

1 Ensemble Learning

We investigated an ensemble learning approach to image classification on TinyImageNet. The 200 image classes were split into five "tasks". For each task, a ResNet-18 was trained for 50 epochs. Training and validation graphs are shown in Figure 1. Validation accuracy on task-specific classes, as well as the combined set comprising all classes, are shown below.

Task	Task Validation Accuracy	Combined Evaluation Accuracy
1	0.71	0.15
2	0.67	0.20
3	0.73	0.07
4	0.67	0.15
5	0.69	0.12

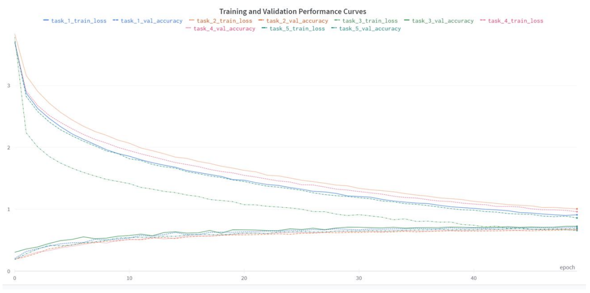

Figure 1: Training and validation performance curves for the ensemble learning experiment using the default task split provided.

Individual task model performances on their respective validation sets are far higher than those for the combined validation set, as the latter contains many more classes which the model was not trained on. This is supported by the comparatively large drop in task 3, from 0.73 to 0.07; task 3 is the task with the least number of classes assigned to it.

To investigate the effect of task split overlap, an overlapping task split was created from the old task split by taking 20% of the classes from each split and distributing them among all tasks, in addition to the pre-existing tasks. For instance, 20% of the classes assigned to task 1 were also made available to to task 2, another 20% of classes from task 1 were appended to the list for task 3, and so on for tasks 4 and 5.

Ensembles were then combined into a single prediction through five methods: random selection, oracle selection, confidence-based selection, entropy-based selection, and stacking. In entropy-based selection, the model outputting the probability distribution with the lowest entropy is chosen. In stacking, 20% of each task's data is withheld for training a classifier head which takes in feature maps outputted from each of the models and outputs a probability distribution for the odds of the image being each of the two hundred classes. Results are reported in Table 1.

In the selection and aggregation schemes without overlap, random selection can be considered a lower bound–the odds of choosing the model trained for that image are 1 in 5. On the other hand, oracle selection, in which the model trained for the image is always chosen, can be taken as an upper bound. Relative to those two scores (0.14 and 0.69, respectively), confidence selection, entropy selection, and stacking all perform relatively similarly with an accuracy of approximately 0.40. Confidence and entropy selection have the same accuracy, possibly because of a correlation–perhaps the most confident model also tends to produce a probability distribution with the smallest entropy. When training a classifier head to amalgamate information from all feature heads, the ensemble performs slightly worse, probably because the correct model offers more accurate predictions than the models trained on other data subsets.

In the case with overlap, random selection does better relative to the case without overlap, as the probability of randomly choosing a model trained on any one class is higher. In contrast, the accuracy

	Without Overlap	**With Overlap**
Random Selection	0.14	0.25
Oracle Selection	0.69	0.64
Confidence Selection	0.43	0.53
Entropy Selection	0.43	0.53
Stacking	0.39	0.52

Table 1: Ensemble combined evaluation set accuracy using different selection and aggregation schemes, as well as different task splits.

of oracle selection decreases slightly, even though the same model is chosen as the "without overlap" case. As each model was trained on more classes, the oracle model may be slightly less specialized to its task split classes without overlap. In contrast, confidence selection, entropy selection, and stacking all increase in accuracy, perhaps because each model was allowed to trained on more data overall. However, their accuracies still fall below that of oracle selection, as the wrong model is still chosen some of the time. Of note, confidence and entropy selection still retain the same accuracy under this new task split. Additionally, stacking still performs a bit worse, but on the same level as confidence and entropy selection. However, the performance drop is less than in the "without overlap" case, as the average quality of prediction from each model has increased.

We observe that our new task split is better than the one provided since we observe higher accuracies obtained. To create effective task splits we suggest that tasks be generally grouped together according to subjective similarity, with tasks that are similar to many other tasks used as the tasks to overlap between groups. It may also be important to ensure particularly difficult or unique tasks are isolated to one group, to ensure that only one model can specialize on that task and the difficult task does not interfere with other models learning general concepts that are too different from the difficult task.

Regarding the way the observations from ensemble learning relate to multi-task learning and transfer learning as well as the key differences between these approaches, we observed that each model in the ensemble is trained independently on a random subset of the data or using different algorithms. The key idea is that the ensemble can capture different aspects of the data and combine them to make better predictions, whereas in multi-task learning, the model shares the same hidden layers across all tasks but has different output layers for each task. in other words, the shared layers learn features that are useful for all tasks, while the task-specific layers learn features that are specific to each task. In transfer learning, the idea is that the features learned from the source task or dataset can be transferred to the target task or dataset, reducing the amount of data needed to train the model and improving its generalization performance. The key differences between these approaches lie in their goals and assumptions. Ensemble learning aims to improve the accuracy and robustness of a model by combining multiple models; on the other hand, Multi-task learning aims to improve the performance of multiple related tasks by sharing information between them. In contrast, transfer learning aims to improve the performance of a target task by transferring knowledge learned from a source task or dataset.

2 Federated Learning

All results were collected using our slightly-modified version of FL-bench, with ResNet-18 clients and the Tiny ImageNet dataset.

2.1 Analysis

2.1.1 Client number and joining ratio

We carried out experiments over a range of join ratios and client numbers. The results for the join ratio experiments are summarized in Figure 4. We tested six join ratio values of 0.1, 0.2, 0.4, 0.6, 0.8, and 1.0, with 100 clients using FedAvg under the IID partition scheme, over 100 global epochs, 5 local epochs, and 1 fine-tuning epoch. We observe that a higher join ratio creates a smoother learning curve, but does not significantly impact the final global model accuracy. From this we conclude that we do not prefer a particularly high or low join ratio in general.

Figure 3 shows the results of our client number experiment, for values of 20, 40, 80, 120, 160, and

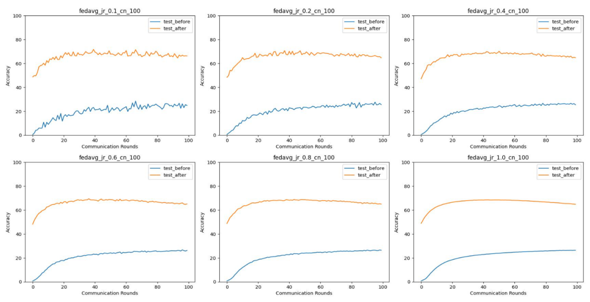

Figure 2: Validation accuracy curves for FedAvg join ratio experiments, showing accuracy before and after fine-tuning.

200, using FedAvg with the same hyperparameters as above except using 50 global epochs. We observe again that raising the client number increases the smoothness of the curve. However, it also slightly changes the final accuracy, with higher client numbers mildly decreasing the final global model accuracy before and after fine-tuning. So, choosing the client number is a trade-off between stability and accuracy.

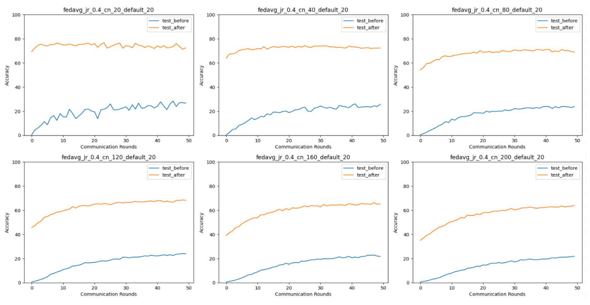

Figure 3: Validation accuracy curves for FedAvg client number experiments, showing accuracy before and after fine-tuning.

2.1.2 Statistical Heterogeneity

We can measure the statistical heterogeneity across clients by examining the non-IID partition hyperparameter. For example, under the shard-based partition, increasing the number of shards per client decreases heterogeneity. A more sophisticated measure of heterogeneity might compute the average earth-mover distance between sampled pairs of client class distributions. For all non-IID partitions, we found that as heterogeneity decreased, final model performance before fine-tuning increased, but performance after fine-tuning decreased. For example, for shards, with 100 clients, increasing the number of shards per client from 2 to 128 increased global pre-fine-tuning accuracy from 5% to 35% and decreased post-fine-tuning global accuracy from 94% to 43%.

To investigate how to handle variable training dataset sizes across clients, we modified the partition code for shard partitioning to include a variability parameter which specifies a upper and lower bound on the randomized number of shards for each client. We also modified the FedAvg code to include a parameter that sets the client-wise global update weights to be either all equal or proportional to the training dataset size, in order to experiment with using weighting to compensate for heterogeneity. We found that for highly-variable numbers of examples across clients, allowing unequal weighting across clients improves the final global accuracy pre-fine-tuning by as much as 2%, but worsens post-fine-tuning accuracy by the same amount.

2.1.3 Local and Global Epochs

We chose 10 random clients from the 40 clients and used the mean, variance and standard deviation of their local learning curves to compare it against the global accuracy on every round.

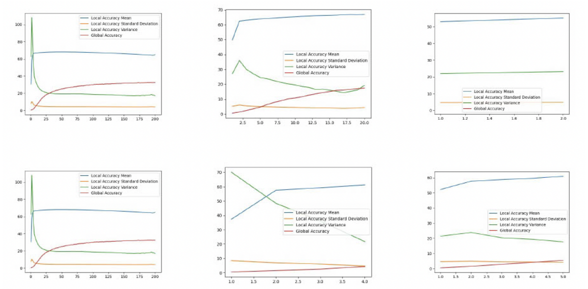

Figure 4: Learning curves. The local learning rate for first row is 0.01 and for second row is 0.1. The global epochs increase from left to right(total epochs is 200).

When the personalization accuracy increases, the global model performs badly because the local models tend to overfit the local data they are trained on. Similarly, the generalization increases along with increasing communication. This is intuitive since increasing the global epochs will contribute to generalizing to the data. Therefore, the best case possible is to keep the keep local epochs high and keep the global epochs relatively higher. This increases the cost drastically. According to our observations, we could dynamically change the number of local epochs depending upon the measured divergence between local and global epochs, or through cross-validation. But this method only increases the communication cost per round. Additionally, we can fix the schedule to change the number of local epochs per round based on some pre-defined rules, such as decreasing the local epochs when the client accuracies reach a pre-defined limit. As mentioned earlier, variable local epochs increases the cost of communication unless we use a pre-defined rule. However, compromising with the communication cost, we can improve the global efficiency of the model by using adaptive learning rate algorithms like Adagrad or Adam, or other techniques for each local training round. Additionally, we can fix the maximum accuracy for specific clients who meet certain efficiency standards.

2.1.4 Global and Personalization Accuracy in FedProx vs. FedAvg

In Table 2, we show global and local accuracy data comparing FedAvg and FedProx with various regularization coefficient values μ. All cases were performed with shard-partitioned data over 100 clients with 16 shards per client, and a join ratio of 0.4 and 50 global, 5 local, and 1 fine-tuning epoch. Results indicate the best overall performance for global and local accuracy for FedAvg, which is the same as FedProx with $\mu = 0$. Increasing the regularization coefficient was strictly worse for performance, except minimum local accuracy at 0.1. This indicates a lack of an interesting trade-off in performance with this hyperparameters in FedProx. Theoretically, in a setting with variable data counts across clients, locally tuning the regularization coefficient would be analogous to the local weighting by dataset size for FedAvg, and similar to our experiments with client weighting in FedAvg, this should be somewhat useful for improving global accuracy.

	FA	FP μ0	FP μ0.1	FP μ1
Pre-fine G	**22.91**	**22.91**	22.79	21.1
Post-fine G	**68.79**	**68.79**	68.59	66.64
Mean L	**22.997**	22.813	22.738	20.773
Max. L	**31.99**	29.04	28.86	26.65
Median L	**22.89**	22.705	22.515	20.59
Min. L	15.62	17.65	**18.2**	16.91

Table 2: Comparing (L)ocal and (G)lobal Accuracy between FedProx and FedAvg

2.1.5 Comparing Meta-learning and FL

FedAvg and FedProx have very similar settings to Model-Agnostic Meta Learning (MAML), since they all consist of iteratively training multiple models over different tasks or class distributions and then combining the weight updates to update the global model or meta model. The local epochs in FL are analogous to the inner loop of MAML, and the global aggregation step is analogous to the Outer loop of MAML, where in MAML the initialization model learns to create a good weight initialization for unseen tasks while in FL the global model learns to do well on all classes using just the weight changes from the client models. The difference is that in MAML the meta-model is trying to learn how to initialize weights for unseen tasks while in FL the global model is trying to learn directly to be good at the task that all the clients learn.

2.1.6 Performance Impact of Local Synchronization in FL

By comparing our ensemble and federated learning results, we can see that for a similar number of epochs, we can see that final model performance before fine-tuning is worse for Federated Learning, but after fine-tuning Federated Learning is superior, with or without overlapping task splits. This suggests that the communication between clients in FL allows the global model to ultimately reach a more "generalizable" point on the loss landscape. While without any fine-tuning, the model performance worse, with one round of fine-tuning we achieve better performance which suggest the location on the loss landscape finally reached in FL ends up proximal to a superior point on the loss landscape compared to Ensemble Learning.

2.1.7 Training an Ensemble Model with FL

Theoretically, one could use federated learning to train an ensemble model by distributing the members of the ensemble across devices. Rather than updating the parameters of a global model, the aggregation step would need to learn a way to combine the outputs of the client models without seeing the client data. We propose synthetically generating examples for training the combiner or gate model in the global step. The global training only needs to select a specialized expert, so it can use less-realistic synthetic data to make that choice while relying on the expert model for realistic accuracy. However, synthetically-generated data might harm accuracy, and an attacker could reverse-engineer real examples using the synthetic examples to violate the security constraint of FL.