Kennesaw State University

College of Computing and Software Engineering

CS 4850: Senior Project

XZ-1 Federated Learning

Aaron Cummings

Justin Bull

Drew Hutchison

Professors

Sharon Perry

Xinyue Zhang

Table of Contents

Introduction

Initial Research

Our initial research was proposed by Dr Xinyue Zhang as a topic in Federated Learning. To begin, we conducted a literature review of the most prominent research papers in the field, and held peer review sessions to communicate what each member learned that week. Doing this allowed us to get much more familiar with the field in order to prepare us for attempting to implement something

Federated Learning

The idea of splitting data and utilizing distributed computing for accelerating model training has been an idea for a while. However, it didn't have a real term for it until Google coined the term 'Federated Learning' (FL) in their paper [1].
FL allows for this accelerated learning by having a foundational model, pre-trained on some data, distributed to each edge device, and having the edge device train that model on their own data, summarize and encrypt that model, reintegrate the new model into the centralized model for averaging and redistribution. This allows for much lower communication costs, elevated data privacy, and accelerated training, due to only the model updates being sent, the data staying on the local network, and allowing multiple devices to train in a distributed manner [2].

Federated learning also provides the architecture to allow for clients on participating edge devices to have their own distinct models, personalized for the local dataset. Since the training is offloaded to a device that may have its own unique distribution of data, the device is able to continue training its local model on this subset of data for more specifically tailored models.

Knowledge Distillation

Knowledge Distillation is an idea within Federated Learning to train on a smaller model, so that the computational power required to train is lowered, without a major tradeoff in performance. In [3], the idea of having a larger mentor, or teacher model, can be used in conjunction with a smaller mentee, or student model, as a way to distill knowledge through the Protégé Effect. Current models are massive and require a lot of computational power and incur a high communication cost to train and communicate, using the smaller model for the heavy lifting, alongside a teacher model that can provide it's results without back propagation, can allow for a higher performing, smaller model that is easier to communicate.

The application of Knowledge Distillation as a training method in federated learning also serves as a method to create federated systems with relatively lower communication costs. In our survey of the field, a major issue with implementing federated systems is the communication overhead. There are many problems that fix the paradigm of a federated learning solution, however exist in systems where communication bandwidth may be limited. By utilizing

Knowledge distillation, you can not only reduce the training costs put on participating clients on edge devices, but also reduce the size of the gradient needed to be transmitted for a round of federated learning. It has been found in [XX] that in combination with gradient compression techniques, this allows federated learning to be an effective solution in such cases.

Choosing a Problem

Federated Learning has many areas of ongoing study to improve the paradigm in different ways, such as, data privacy, efficiency, communication, and reliability. Our motivation was centered around finding a widely common problem with implementations of federated learning models, data heterogeneity. An important consideration as well was addressing the problem while maintaining many of the most recent improvements that have been made to federated learning.

Federated learning strategies vary widely, the method in which client models are selected and combined is an important consideration for the use of federated learning, each having its benefits and costs. Hence, the implementation used assumed that the federated model would need efficient computation and communication costs. Taking inspiration from generative federated learning strategies, and using leading methods of effective federated learning lead to the implementation discussed in this paper.

Data Heterogeneity

Data heterogeneity is a major concern in federated learning. A normal assumption when training a model is that the data is i.i.d, independent and identically distributed, this assumption does not hold in many federated learning settings. Data heterogeneity leads to poor model performance and difficulty training. There are several ways that data heterogeneity can be present. The amount of data on each client could vary in quantity, for example the number of patients at a hospital. Data format or quality can also vary, one hospital may provide higher resolution imaging than another. The distribution of data can also vary by client, one hospital may have a widely different demographic from another such as elderly who disproportionately suffer from different health conditions from younger demographics. And finally the features of the data can also vary, in a hospital different tests might be run to indicate the same condition.

In federated learning a single client who has a smaller, vastly different data distribution or features may suffer from poor model performance. Alternatively another client that has vast amounts of data that is dramatically different in distribution or features will cause the shared trained model to perform poorly on all other client's data by outweighing the impact of their training or participating in training more frequently.

Implementation

The implementation portion of our project aims to address the problem we identified in our research. Each member of the was tasked with implementing one of the following, Federated Learning, Knowledge Distillation, and Variational Autoencoder. Each functional component can then be combined to address the problem of data heterogeneity in a federated learning model that utilizes the techniques discovered in our research to improve federated learning.

Dataset

MedMNIST

The MedMNIST Dataset was chosen as a viable dataset to simulate data heterogeneity. An application found in research where federated learning could be applied and suffer from data heterogeneity was hospital data. Patient data is private data that is stored in secure networks that would likely not be provided to be aggregated by a third party in a centralized learning model. Hospitals also vary in equipment used to gather data.

Implementation Tools

Flower

The network of clients participating in federated learning and the central server aggregating model updates was simulated using Flower. Flower is an open source python project that allows for both simulation of federated systems and the creation of deployable clients. For the purposes and resource limitations of this work, simulating the network was an important step for implementation. The following is a modified version of the Flower quickstart tutorial for demonstration and explanation.

Model Parameter functions: (PyTorch)

```
def get_parameters(net) -> List[np.ndarray]:
    return [val.cpu().numpy() for _, val in net.state_dict().items()]

def set_parameters(net, parameters: List[np.ndarray]):
    params_dict = zip(net.state_dict().keys(), parameters)
    state_dict = OrderedDict({k: torch.Tensor(v) for k, v in params_dict})
    net.load_state_dict(state_dict, strict=True)
```

In order to implement a federated learning model with Flower, there are only a few classes and methods that need to be implemented. Functions for retrieving and setting model parameters are given by Flower, and need to be overwritten for either PyTorch or TensorFLow and use numpy for manipulation. Flower is capable of using either libraries for the model definition and acts as a wrapper, which provides a significant level of abstraction from the implementation of the model used on the clients.

Defining a model:

```
class Net(torch.nn.Module):
    def __init__(self) -> None:
        # Model Definition

    def forward(self, x: torch.Tensor) -> torch.Tensor:
       # Forward Propagation
        return output
```

The model itself can then be defined as if you were implementing a class containing the model for a monolithic learning model as well as training and testing functions to provide the forward and backward passes over a given number of epochs.

Training and Testing Functions:

```
def train(net, trainloader, epochs: int, verbose=False):
    """Train the network on the training set."""
    net.train()
    for epoch in range(epochs):
        for images, labels in trainloader:
            # Forward and Backward Step
            # Define Metrics
        if verbose:
            print(f"Epoch {epoch+1}: train loss {epoch_loss}, accuracy {epoch_acc}")


def test(net, testloader):
    """Evaluate the network on the entire test set."""
    return loss, accuracy
```

The clients then need to be defined using Flower, which extends the NumPyClient class with several methods needed. Torch Trainloaders are an effective method for managing data in an iterable manner.

Flower Client:

```
class FlowerClient(fl.client.NumPyClient):
    def __init__(self, net, trainloader, valloader):
        self.net = net
        self.trainloader = trainloader
        self.valloader = valloader
```

```
    def get_parameters(self, config):
        return get_parameters(self.net)

    def fit(self, parameters, config):
        set_parameters(self.net, parameters)
        train(self.net, self.trainloader, epochs=1)
        return get_parameters(self.net), len(self.trainloader), {}

    def evaluate(self, parameters, config):
        set_parameters(self.net, parameters)
        loss, accuracy = test(self.net, self.valloader)
        return float(loss), len(self.valloader), {"accuracy": float(accuracy)}
```

Flower requires that you implement the function client_fn that allows the simulation to create clients. Partitioning the dataset into a list of distinct sets of torch DataLoaders is an effective method for distributing unique data to each client.

Flower client_fn for client simulation:

```
def client_fn(cid: str) -> FlowerClient:
    """Create a Flower client representing a single organization."""

    # Load model
    net = Net().to(DEVICE)

    # Load data
    # Note: each client gets a different trainloader/valloader, so each client
    # will train and evaluate on their own unique data
    trainloader = trainloaders[int(cid)]
    valloader = valloaders[int(cid)]

    # Create a  single Flower client representing a single organization
    return FlowerClient(net, trainloader, valloader)
```

Flower includes a way to describe your own metrics for the federated system.

Evaluation Metrics:

```
def weighted_average(metrics: List[Tuple[int, Metrics]]) -> Metrics:
    # Multiply accuracy of each client by the number of examples used
    accuracies = [num_examples * m["accuracy"] for num_examples, m in metrics]
    examples = [num_examples for num_examples, _ in metrics]

    # Aggregate and return custom metric (weighted average)
    return {"accuracy": sum(accuracies) / sum(examples)}
```

Flower comes packaged with many strategies for the simulated server to operate with for model update aggregation, however you may define your own. The packaged strategies come with several parameters.

Strategy:

```
# Create FedAvg strategy
strategy = fl.server.strategy.FedAvg(
    fraction_fit=1.0,  # Sample 100% of available clients for training
    fraction_evaluate=0.5,  # Sample 50% of available clients for evaluation
    min_fit_clients=10,  # Never sample less than 10 clients for training
    min_evaluate_clients=5,  # Never sample less than 5 clients for evaluation
    min_available_clients=10,  # Wait until all 10 clients are available
    evaluate_metrics_aggregation_fn=weighted_average,  # <-- pass the metric aggregation
function
)
```

The federated simulation can then be started by calling the start_simulation method.

Starting simulated Federated Learning:

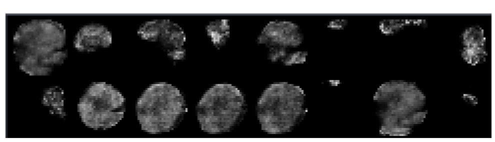
```
# Start simulation
fl.simulation.start_simulation(
    client_fn=client_fn,
    num_clients=NUM_CLIENTS,
    config=fl.server.ServerConfig(num_rounds=5),
    strategy=strategy,
    client_resources=client_resources,
)

# Sample Output
History (loss, distributed):
        round 1: 31.653484654426574
        round 2: 27.158826851844786
        round 3: 25.561719250679015
        round 4: 24.670892000198364
        round 5: 23.940561509132387
History (metrics, distributed):
{'accuracy': [(1, 0.294), (2, 0.37200000000000005), (3, 0.4096), (4, 0.43), (5,
0.44959999999999994)]}
```

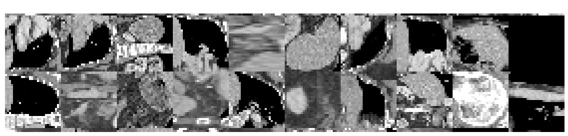
VAE

The autoencoder is a method of generating new data when another client is under-performing as a way of mitigating data heterogeneity. This is done through a Variational Autoencoder consisting of the following architecture built from PyTorch:

```python
class VAE(nn.Module):
    def __init__(self, x_dim, h_dim1, h_dim2, z_dim):
        super(VAE, self).__init__()

        # Encoder
        self.fc1 = nn.Linear(x_dim, h_dim1)
        self.fc2 = nn.Linear(h_dim1, h_dim2)
        self.fc31 = nn.Linear(h_dim2, z_dim)
        self.fc32 = nn.Linear(h_dim2, z_dim)
        # Decoder
        self.fc4 = nn.Linear(z_dim, h_dim2)
        self.fc5 = nn.Linear(h_dim2, h_dim1)
        self.fc6 = nn.Linear(h_dim1, x_dim)

    def encoder(self, x):
        h = F.relu(self.fc1(x))
        h = F.relu(self.fc2(h))
        return self.fc31(h), self.fc32(h) # mu, log_var

    def sampling(self, mu, log_var):
        std = torch.exp(0.5*log_var)
        eps = torch.randn_like(std)
        return eps.mul(std).add_(mu) # return z sample

    def decoder(self, z):
        h = F.relu(self.fc4(z))
        h = F.relu(self.fc5(h))
        return F.sigmoid(self.fc6(h))

    def forward(self, x):
        mu, log_var = self.encoder(x.view(-1, 784))
        z = self.sampling(mu, log_var)
        return self.decoder(z), mu, log_var
```

After training this on a sample set from MedMNIST, specifically Organ CT Scans, it was able to generate relatively similar data from a random distribution.





An advantage of a VAE over a normal AE, is that instead of just replicating an image that has already existed, we can generate new data from a random sample of the latent space that we can define. The VAE will have never seen the vectors generated from the random sampling, letting us have new data to work with.

Knowledge Distillation

Knowledge Distillation is a method of compression for a machine learning modell. To perform Knowledge Distillation you need a pretrained model named the teacher model and a smaller untrained model named the student. The intent is to take the knowledge from the pretrained teacher model and distill it into the smaller student model. With a quality teacher model, Knowledge Distillation produces a student model that converges faster and with higher accuracy than an identical student model trained without Knowledge Distillation.

Generally during training a model only considers the difference between its guess and the answer. However, knowledge distillation is performed by modifying the loss function of the student to include the difference between its logits (prediction) and the logits of the teacher. So, in knowledge distillation, the student not only considers how far off it’s guess was from the correct answer but also how far it’s guess was from the teacher model’s guess. It’s because of the additional information provided by the teacher during training that allows the student model to converge more quickly and with more accuracy than without the teacher’s logits.

```
def train_kd(student_model, teacher_model, trainloader, temperature, alpha):
    ‘’’ Train student model with pre-trained teacher model’’’

    for epoch in range(epochs):
        for images, labels in trainloader:
            # Forward pass to gain student and teacher predictions
            studentOut = student_model(input)
            teacherOut = teacher_model(input)
            # Calculate the loss between the student and teacher
            distillation_loss = kd_loss_function(studentOut, teacherOut, temperature)
            # Calculate the loss between the student and the correct output
            student_loss = student_loss_function(studentOut)
            # Calculate the total loss using regular loss calculation combined with kd loss
            total_epoch_loss = alpha * student_loss + (1-alpha) * distillation_loss
            # Continue train step as normal, performing backpropagation
            …
```

Here, alpha is the learning rate and temperature is a hyper-parameter used to normalize the teacher and student's predictions prior to calculating the distillation loss. There will be two criterion functions, one for the student loss and another for distillation loss. There are a variety of functions that can be used based on the needs of the programmer, but regardless of what functions are chosen, the distillation process remains the same.

Conclusion

Our project attempted to utilize Federated Learning, Knowledge Distillation, and an Autoencoder to encourage supportive data privacy with the ability to generate new data when one edge device has a data quantity problem.
We were able to create the components and build an ensemble model utilizing the three key features, specifically on a dataset of medical images from the MedMNIST dataset.

Appendix

Literature Reviews Document

Paper/Article	Reviewer	Summary
https://ai.googleblog.com/2017/04/federated-learning-collaborative.html	Aaron Cummings	This article from google about federated learning is an introduction to the concept that was published early in the federated learning exploration. The post explores the different applications of this learning model that google has successfully used, like keyboard suggestions, and discusses potential areas this could be implemented in. It provides a simple run down of the methodology, benefits, and areas of improvement for the concept and ends with a call to action for greater discussion and exploration of the concept from the community.
https://ai.googleblog.com/2017/04/federated-learning-collaborative.html	Andrew Hutchison	This article discusses how Federated Learning works at a high level, and then explains how Google is currently applying Federated Learning into their ‘G-Board’ app via their own Federated Averaging Algorithm. The article also describes how they reduce necessary communication while also maintaining high model quality through their FA algorithm, compression, and the use of specialized algorithms for certain tasks.
Communication efficient federated learning via knowledge distillation	Aaron Cummings	The authors of this paper introduce their own federated learning model called FedKD, where they use knowledge distillation and a large mentor model and smaller mentee model that distill knowledge to each other based on the Protégé Effect. Their reasoning is that in current techniques, the models are too large to effectively communicate. They also submit that by using this method, each client can have a larger more personalized model. They use singular value decomposition based dynamic gradient approximation to compress the communicated models dynamically.

Communication-efficient federated learning via knowledge distillation	Justin Bull	Big Idea: "Learn intelligent models from decentralized private data" Current Problems Introduced: Newer models have become so large in size that the communication overhead is expensive, which is impractical. Generally, larger models aren't used in conventional fed learning systems. Potential Solutions: Gradient Compression | Codistillation Implements a model utilizing a solution they introduced – FedKD – and compared to other industry Federated Learning models.
Data-Free knowledge Distillation for Heterogeneous Federated Learning	Aaron Cummings	This paper introduces the author's concept and implementation of FeDGen, and Knowledge Distillation method to resolve data heterogeneity among clients in a federated learning system. The concept primarily operates on the concept of using local data to create and share a generative model that can then be used to generate data that offsets the heterogeneity of other local models. The generator is trained on the prediction rule of user models, to abstract user data. The generative model data becomes an inductive bias for users that have limited data, or also assume little variety in data. This allows their users to adjust their decision boundaries to approach the ensemble wisdom. *The related work section covers a lot of other methods that would be helpful to review.*
Reliable Federated Learning for Mobile Networks	Aaron Cummings	This paper introduces the concept of reputation to federated learning to distinguish between trusted and reliable workers/clients and unreliable updates. Some of the data that would be considered unreliable is data poisoning that is intentional or unintentional, low-quality data from energy constraints or communication. The approach uses a block chain to decentralize and prevent tampering of reputation management. This process uses consortium blockchains that perform the consensus process on pre-selected miners, which is supposed to be cost and time effective. Federated learning addresses critical challenges of machine learning around single points of failure, data leakage, storage and communication overheads.

Federated Learning: Challenges, Methods, and Future Directions	Aaron Cummings	This paper discusses the current state and projected future paths of Federated Learning, while explaining the core motivations and concepts behind the approach and implementation. Both the applications and the edge devices that could utilize federated learning models are explored and discussed. As well as the issues and benefits of those devices and applications. Explains how federated learning can be used in silo-ed data such as hospitals. The paper has a good explanation of the hypothesis formulation and cost function minimization for Federated learning. Privacy, communication and Heterogeneity are all discussed at length and how they impact Federated Learning. Scale is introduced as a constraint, Local updating schemes where the local gradients are averaged after a variable number of local updates rather than a mini-batch SGD aggregated across edge devices. This approach allows local models to update regularly and cut down on communication overhead. Communication efficiency is decomposed into three groups, local updating methods, compression schemes, and decentralized training. Decentralized training through network topology is presented as an approach that is applicable to some applications of federated learning. Many future directions for research are presented including, extreme communication schemes, communication and reduction and the Pareto frontier, Novel models of asynchrony, Heterogeneity diagnostics, granular privacy constraint, beyond supervised learning, productionizing federated learning, and benchmarks.
https://www.tensorflow.org/federated/federated_learning	Aaron Cummings	TFF is an interface that allows you to experiment with federated learning with an existing model. It also provides datasets for learning. There are tutorials on image classification and text generation.

Federated Learning: Strategies for improving communication efficiency	Aaron Cummings	This paper discusses reducing network communication for a typical client in two ways, structured updates, where learning an update is done from a restricted space parameterized using a smaller number of variables and sketched updates, where a full model update is learned and then it is compressed using quantization, random rotations, and subsampling. The problem is defined as a large number of clients with highly unbalanced non-I.i.d data and poor network connections.
Data-Free Knowledge Distillation for Heterogeneous Federated Learning	Justin Bull	Big Idea: Proposal of a system - FedGen Benefits: Attempts to extract more information out of the limited data availability from the users Improves upon the local models as well as the global models - Better for generalization Communication Costs attempt to be nullified through only sharing the predictive layer of the local models rather than the entire model param Similar to Communication-efficient federated learning via knowledge distillation, proposes a model, and implements and analyzes the results. Has a lot more math detailed/explained within the paper itself.
What is Federated Learning? ODSC - Open Data Science [Link]	Justin Bull	Explains the main idea of what Federated Learning is – training a local and global model while attempting to utilize data privacy measures before redistributing a better model to the users. Benefits Established: Real Time predictions Collaborative training on a range of data sources Privacy through keeping data on local devices No need for internet connection on predictions Attempts to use the cluster idea to remove some of the necessary hardware requirements Similar to the First two papers: There is a problem involving minimal data availability that must be addressed

A Survey on Federated Learning for Resource-Constrained IoT Devices	Andrew Hutchison	This paper provides a literature review on existing Federated Learning studies and then explains we might train distributed ML models on, or using, resource-constrained Internet of Things devices. The benefits of FL are described as well as some of the complications/bottlenecks that come with the implementation of FL. Then, an overview of several FL survey papers is given and a brief comparison of different types of FL models is given. The FL models covered includes Horizontal FL, Vertical FL, and Federated transfer learning. Furthermore, the paper gives a high-level overview on 4 different Federated Learning Algorithms.
Federated Learning: A Survey on Enabling Technologies, Protocols, and Applications	Justin Bull	Survey Paper Doesn't add much more than https://odsc.medium.com/what-is-federated-learning-99c7fc9bc4f5 or Data-Free Knowledge Distillation for Heterogeneous Federated Learning Does include a summary of Optimization techniques and best practices for Federated Learning Systems Introduces an idea for data stability and helping lower communication costs through choosing reputable and trusted local models 'mentees' (term is used in a different paper with a similar system of mentors and mentees), and only gathering data from a subset of the local models at a time – depending on how trustworthy they are evaluated to be.
Robust and Resource-Efficient Data-Free Knowledge Distillation by Generative Pseudo Replay	Aaron Cummings	This paper expands on data-free KD, where an encoder and decoder are trained and used for knowledge distillation and data synthesis. They explore the idea of using generative replay strategies in continuous learning to prevent catastrophic forgetting.

References

[1] B. McMahan and D. Ramage, "Federated learning: Collaborative machine learning without centralized training data," Google AI Blog, 06-Apr-2017. [Online]. Available: https://ai.googleblog.com/2017/04/federated-learning-collaborative.html. [Accessed: 29-Apr-2023].

[2] K. Martineau, “What is federated learning?,” *IBM Research Blog*, 24-Aug-2022. [Online]. Available: https://research.ibm.com/blog/what-is-federated-learning. [Accessed: 29-Apr-2023].

[3] C. Wu, F. Wu, L. Lyu, Y. Huang, and X. Xie, “FedKD: Communication Efficient Federated Learning via knowledge distillation,” arXiv.org, 24-Apr-2022. [Online]. Available: https://arxiv.org/abs/2108.13323. [Accessed: 29-Apr-2023].

[4] Z. Zhu, J. Hong, and J. Zhou, “Data-free knowledge distillation for heterogeneous federated learning,” arXiv.org, 09-Jun-2021. [Online]. Available: https://arxiv.org/pdf/2105.10056.pdf. [Accessed: 29-Apr-2023].

[5] J. Kang, Z. Xiong, D. Niyato, Y. Zou, Y. Zhang, and M. Guizani, “Reliable Federated Learning for Mobile Networks,” *IEEE Wireless Communications*, vol. 27, no. 3, pp. 72–80, Feb. 2020.

[6] T. Li, A. K. Sahu, A. Talwalkar, and V. Smith, “Federated learning: Challenges, methods, and future directions,” *IEEE Signal Processing Magazine*, vol. 37, no. 3, pp. 50–60, May 2020.

[7] “Federated learning ,” *TensorFlow*. [Online]. Available: https://www.tensorflow.org/federated/federated_learning. [Accessed: 29-Apr-2023].

[8] J. Konečný, H. B. McMahan, F. X. Yu, P. Richtárik, A. T. Suresh, and D. Bacon, “Federated learning: Strategies for improving communication efficiency,” *arXiv.org*, 30-Oct-2017. [Online]. Available: https://arxiv.org/abs/1610.05492. [Accessed: 29-Apr-2023].

[9] O. D. S. C.- O. D. Science, “What is federated learning?,” *Medium*, 03-Apr-2020. [Online]. Available: https://odsc.medium.com/what-is-federated-learning-99c7fc9bc4f5. [Accessed: 29-Apr-2023].

[10] A. Imteaj, U. Thakker, S. Wang, J. Li, and M. H. Amini, “A survey on Federated Learning for Resource-constrained IOT devices,” *IEEE Internet of Things Journal*, vol. 9, no. 1, pp. 1–24, Jul. 2022.

[11] M. Aledhari, R. Razzak, R. M. Parizi, and F. Saeed, “Federated learning: A survey on enabling technologies, Protocols, and applications,” *IEEE Access*, vol. 8, pp. 140699–140725, Jul. 2020.

[12] K. Binici, S. Aggarwal, N. T. Pham, K. Leman, and T. Mitra, “Robust and resource-efficient data-free knowledge distillation by Generative Pseudo Replay,” *arXiv.org*, 21-Mar-2022. [Online]. Available: https://arxiv.org/abs/2201.03019. [Accessed: 29-Apr-2023].

[13] Jiancheng Yang, Rui Shi, Bingbing Ni. "MedMNIST Classification Decathlon: A Lightweight AutoML Benchmark for Medical Image Analysis". IEEE 18th International Symposium on Biomedical Imaging (ISBI), 2021.

[14] Jiancheng Yang, Rui Shi, Donglai Wei, Zequan Liu, Lin Zhao, Bilian Ke, Hanspeter Pfister, Bingbing Ni. Yang, Jiancheng, et al. "MedMNIST v2-A large-scale lightweight benchmark for 2D and 3D biomedical image classification." Scientific Data, 2023.

[15] D. C. Nguyen, M. Ding, P. N. Pathirana, A. Seneviratne, J. Li, and H. V. Poor, “Federated learning for internet of things: A comprehensive survey,” *arXiv.org*, 16-Apr-2021. [Online]. Available: https://arxiv.org/abs/2104.07914. [Accessed: 29-Apr-2023].

[16] H. Shah, “Knowledge distillation for convolution neural networks using Pytorch,” *Het Shah | Knowledge Distillation for Convolution Neural Networks using Pytorch*, 16-Mar-202AD. [Online]. Available: https://het-shah.github.io/blog/2020/Knowledge-Distillation/. [Accessed: 29-Apr-2023].