5-10-2024

Distributed System for MeshNet

Pratyush Reddy Gaggenapalli

Distributed System for MeshNet

by

Pratyush Reddy Gaggenapalli

Under the Direction of Sergey Plis Ph.D.

A Thesis Submitted in Partial Fulfillment of the Requirements for the Degree of

Master of Science

in the College of Arts and Sciences

Georgia State University

2024

ABSTRACT

This thesis explores the development of distributed systems for image classification using deep learning techniques. The primary focus is on optimizing the training of a MESHNET convolutional neural network (CNN) model for accurate image classification. We address the challenges associated with distributed training by proposing novel strategies that maximize resource utilization while maintaining high classification accuracy with minimal data and limited epochs.

To facilitate distributed training, we leverage the Coinstac platform, enabling seamless coordination and computation across distributed nodes. Within this framework, we develop a computation for the MESHNET model, allowing efficient utilization of local data resources for model training. Additionally, we introduce Immunetworks, a lightweight RESTful framework, for scalable deployment and management of distributed systems.

We demonstrate the effectiveness of our approach in achieving high accuracy in image classification tasks while efficiently utilizing distributed resources. Our findings not only contribute to the advancement of distributed deep learning methodologies but also hold promise for a wide range of real-world applications beyond image classification.

INDEX WORDS: MeshNet, Convolutional Neural Network, brain tissue segmentation, medical image analysis, distributed learning, resource optimization

Distributed System for MeshNet

by

Pratyush Reddy Gaggenapalli

Committee Chair: Sergey Plis

Committee: Yanqing Zhang

Harshvardhan Gazula

Electronic Version Approved:

Office of Graduate Services

College of Arts and Sciences

Georgia State University

May 2024

DEDICATION

I dedicate this work to my loving family, whose unwavering support and encouragement have been my guiding light throughout this journey. I am deeply grateful for their sacrifices and belief in my abilities.

I would also like to express my heartfelt appreciation to my advisor, Dr. Sergy Plis, for his invaluable guidance, wisdom, and unwavering support. His expertise and encouragement have been instrumental in shaping this research.

Furthermore, I extend my sincere gratitude to my mentor, Mohamad Masoud, for his insightful feedback, encouragement, and dedication. His mentorship has been invaluable in my personal and academic growth.

This work is dedicated to all those who have inspired and supported me along the way.

ACKNOWLEDGMENTS

I would like to express my sincere gratitude to my committee members for their invaluable feedback and support throughout the development of this document. Their expertise and guidance have been instrumental in shaping the direction of this research.

I am deeply thankful to my instructors for their dedication to teaching and their encouragement of my academic pursuits. Their passion for knowledge has inspired me to strive for excellence in my studies.

TABLE OF CONTENTS

LIST OF FIGURES

CHAPTER 1
INTRODUCTION

1.1 Background and Motivation

Meshnet (4; 8; 5) a specialized deep learning model, holds promise in enhancing our understanding of MRI brain scans. With its ability to meticulously analyze image details, Meshnet can accurately identify different brain tissues like gray matter and white matter. This precision is crucial for tasks such as disease diagnosis and treatment planning, where accuracy is paramount.

In our research, we're also leveraging distributed learning, a collaborative approach where multiple computers, each with its own dataset, work together. Rather than sharing raw data between nodes, which can raise privacy concerns, each node processes its own data and computes gradients—the mathematical directions for improving the model's performance. These gradients are then centrally aggregated, resulting in a refined model that is sent back to each node for further training.

By integrating Meshnet (4; 8; 5) with distributed learning , we aim to streamline the analysis of brain scans while respecting data privacy. This combination allows for faster and more efficient model training, utilizing the collective power of distributed resources. Ultimately, our goal is to empower medical professionals and researchers with enhanced tools for accurately interpreting MRI scans (3), leading to earlier detection of brain abnormalities and deeper insights into brain function and pathology.

1.2 Research Objectives and Contributions

The thesis is focused on achieving the following objectives:

- We aim to develop an optimized Meshnet (4; 8; 5) deep learning model tailored for accurate segmentation of brain tissues in MRI scans (3). This involves refining the model architecture to capture intricate spatial dependencies and enhance segmentation accuracy.

- Investigate distributed learning strategies to enable collaborative model training across multiple nodes. This exploration includes approaches such as centralized aggregation and serverless architectures (13; 14), aiming to improve training efficiency while maintaining data privacy. In particular, our focus extends to gradient aggregation methods (8) within distributed learning frameworks, ensuring seamless collaboration among nodes while preserving data integrity.

- we integrate distributed computing frameworks like Coinstac (12; 1) to streamline centralized gradient aggregation from diverse nodes. This integration enables collaborative model training without compromising data security. Furthermore, we explore the feasibility of serverless architectures (13), leveraging technologies like AWS Lambda and API Gateway, to ensure data security while enhancing scalability and cost-effectiveness in gradient aggregation.

1.3 Thesis Outline

This thesis explores the optimization of brain tissue segmentation in MRI scans. Chapter 2 provides an explanation of Meshnet (4; 8; 5) architecture. In Chapter 3, we present an optimized Meshnet model for accurate segmentation and its results. Chapter 4 discusses distributed learning strategies for collaborative model training. Chapter 5 concludes by summarizing contributions and suggesting future research directions.

CHAPTER 2
ARCHITECTURE OF MESHNET

2.1 MeshNet design

MeshNet (4; 8; 5) leverages a three-dimensional Convolutional Neural Network (3D CNN) architecture. This design allows the model to capture spatial dependencies and relationships in three dimensions, making it particularly suited for tasks like brain tissue segmentation where the 3D nature (3) of the data is essential.

2.1.1 Eight-Layer Design

The architecture 2.1 comprises eight layers, each contributing to the overall feature extraction and segmentation process. The layered structure allows the network to progressively learn hierarchical representations of the input data.

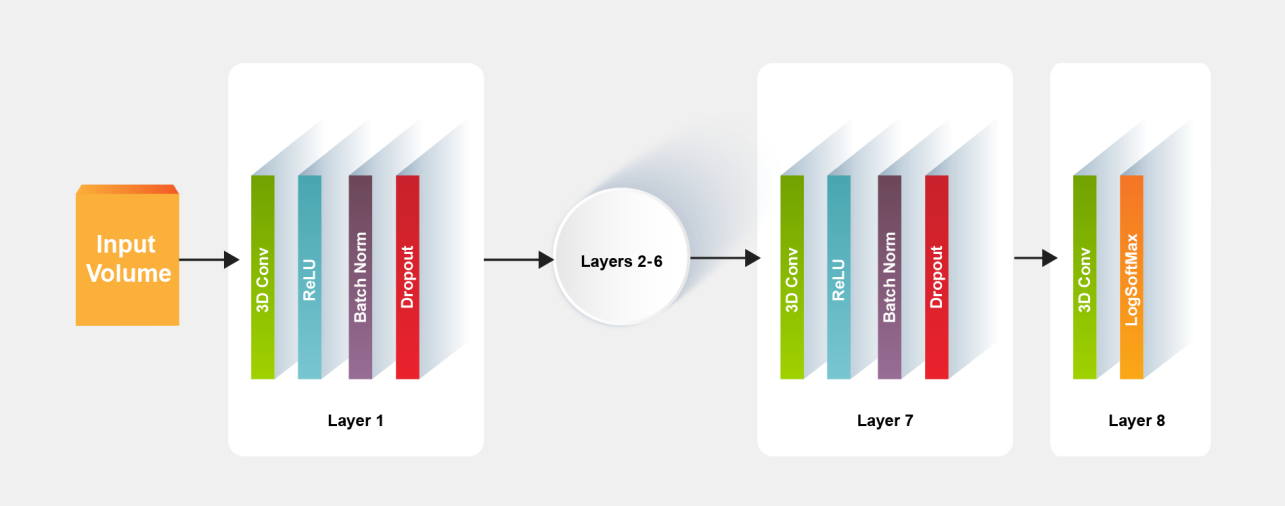


Figure 2.1 MeshNet Architecture (4)

2.1.2 3D Dilated Convolutions

A key feature of MeshNet is the incorporation of 3D dilated convolutions (7). Dilated convolutions, also known as atrous convolutions, enable the network to increase its receptive field without significantly increasing the number of parameters. This is beneficial for capturing long-range dependencies in the input data (3).

2.1.3 Specific Padding Setting and Dilation Factor

Each layer in MeshNet incorporates 3D dilated convolutions (7) with a specific padding setting and dilation factor. These parameters are carefully chosen to balance the trade-off between receptive field size and computational efficiency.

2.1.4 Additional Techniques for Performance Enhancement

- **3D Batch Normalization:** Applied in each layer, batch normalization (7) helps in normalizing the input to a layer, mitigating internal covariate shift and accelerating training.
- **ReLU Activation:** Rectified Linear Unit activation functions introduce non-linearity to the model (7), allowing it to learn complex mappings between inputs and outputs.
- **3D Dropout Regularization:** Dropout (7) is employed to prevent overfitting by randomly setting a fraction of input units to zero during training. In MeshNet, this technique is applied in 3D to enhance regularization.

CHAPTER 3
TRAINING IMPLEMENTATION AND RESULTS

In this chapter, we delve into the detailed methodology employed for training the MeshNet model, including the data loading process, subvolume generation, interactive data exploration, metrics used for evaluation, and the training algorithm (12; 14; 13). Furthermore, we present the experimental results obtained from training the MeshNet architecture on diverse subvolume shapes, showcasing its adaptability and efficiency across different datasets.

3.1 Trining Algorithm

3.1.1 Data Loader

The data loader (14; 12) component plays a crucial role in managing volumetric medical imaging data for brain scan analysis. Integrated with an SQLite database, this module streamlines the retrieval and preprocessing of image-label pairs, essential for training convolutional neural networks (CNNs) like MeshNet. By efficiently handling tasks such as decompression and transformation into PyTorch tensors (6; 3), the data loader ensures that the neural network receives properly formatted input data. Additionally, the data loader's capability to partition the data into smaller sub-cubes enhances processing efficiency by breaking down large volumes into manageable chunks, optimizing resource utilization during training and validation phases. This overview highlights the significance of the data loader component in facilitating seamless integration with neural network models, ultimately contributing to the scalability, reproducibility, and efficiency of medical imaging workflows.

3.1.2 Interactive 3D Scan Data Exploration

Interactive 3D exploration is essential for several reasons, particularly in the context of medical imaging and neural network applications like MeshNet (7). It enhances the interpretability of model predictions, facilitates quality assurance, and provides an intuitive means for users to interactively analyze and communicate insights from the volumetric data and model outputs. We have also use Brainchop for visualizing the Images (10; 9).

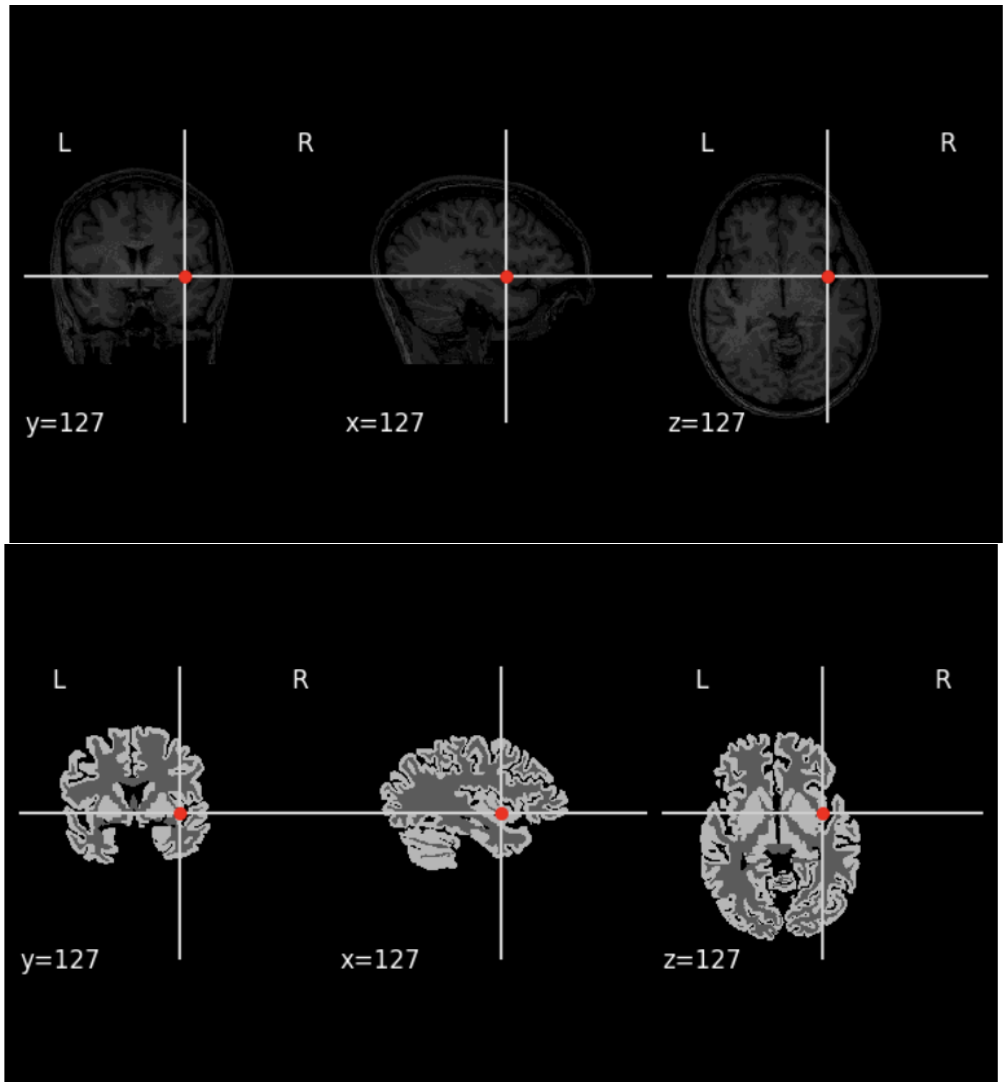


Figure 3.1 Example of interactive 3D scan for input lable and output prediction.

3.1.3 Metrics for assessing model performance

3.1.3.1 Dice coefficient (Dice score)

Dice scores serve as a pivotal metric for quantifying the spatial agreement between predicted and ground truth label maps. Specifically tailored for segmentation tasks, this metric calculates the intersection of pixels in both sets.

$$\text{DICE} = \frac{2|X \cap Y|}{|X| + |Y|}$$

Where:

- X represents the predicted segmentation mask.
- Y represents the ground truth segmentation mask.
- $|X|$ is the number of pixels in the predicted mask.
- $|Y|$ is the number of pixels in the ground truth mask.

The DICE coefficient quantifies the spatial agreement between model predictions (X) and ground truth (Y).

3.1.3.2 Cross Entropy Loss

This measure operates by quantifying the dissimilarity between predicted probabilities assigned by the model and the corresponding true labels.

$$\text{Cross Entropy Loss} = -\sum(y \cdot \log(p))$$

Where:

- y is the true label (one-hot encoded).
- p is the predicted probability distribution.

3.1.4 One-step Learning Rate for Optimizer

In machine learning, the learning rate is a hyperparameter that determines the step size at each iteration while moving toward a minimum of a loss function. One common strategy for adjusting the learning rate during training is the one-step learning rate method.

In this method, the learning rate is typically adjusted after every training epoch or after a certain number of iterations. The adjustment can be based on various factors such as the performance of the model on the validation set or the number of epochs completed.

The one-step learning rate can be implemented using different techniques such as:

- **Step Decay**
- **Exponential Decay**
- **Adaptive Learning Rate (Adam, RMSProp)**

The choice of the one-step learning rate strategy depends on the specific problem, model architecture, and dataset characteristics. Experimentation and tuning are often necessary to find the optimal learning rate schedule for a given task.

3.1.5 Training algorithm:

The training process occurs in the train method, where the model is trained (14; 13) for a specified number of epochs. Within each epoch, the algorithm iterates over the training data, computes the loss and Dice scores, performs backpropagation, updates the learning rate using the one-cycle schedule, and updates the model's parameters using the optimizer. It also evaluates the model on the validation data, calculating the loss and Dice scores for collecting mertics on each cycle.

Listing 3.1 Training Algorithm with One-Cycle LR

```
while epoch != num_epochs:
  model.train()
  train_loss, train_dice = 0.0, 0.0
  for images, labels in trainloader:
      optimizer.zero_grad()
      outputs = model(images)
      loss = criterion(outputs, labels)
      dice_scores = dice(outputs, labels)
      loss.backward()
      optimizer.step()
      scheduler.step()
```

3.2 Results for training

Using Google Colab's GPU infrastructure, our MeshNet underwent training on Mindboggle 101 brain MRI scans, encompassing various sub-volume shapes. The dataset included Brain Scan Datasets with T1 scans and limited labels, partitioned into training (10 pairs), validation (2 pairs), and inference (3 pairs) datasets. MeshNet showcased adaptability across diverse spatial resolutions like 256^3, 32^3, 64^3, and 128^3. Notably, it achieved satisfactory accuracy in fewer training epochs for each sub-volume shape's data-loader, emphasizing its efficiency in extracting features from intricate datasets. This underscores MeshNet's adaptability in multiclaas 3D segmentation tasks within the Mindboggle 101 dataset, portraying its potential for diverse complexities.

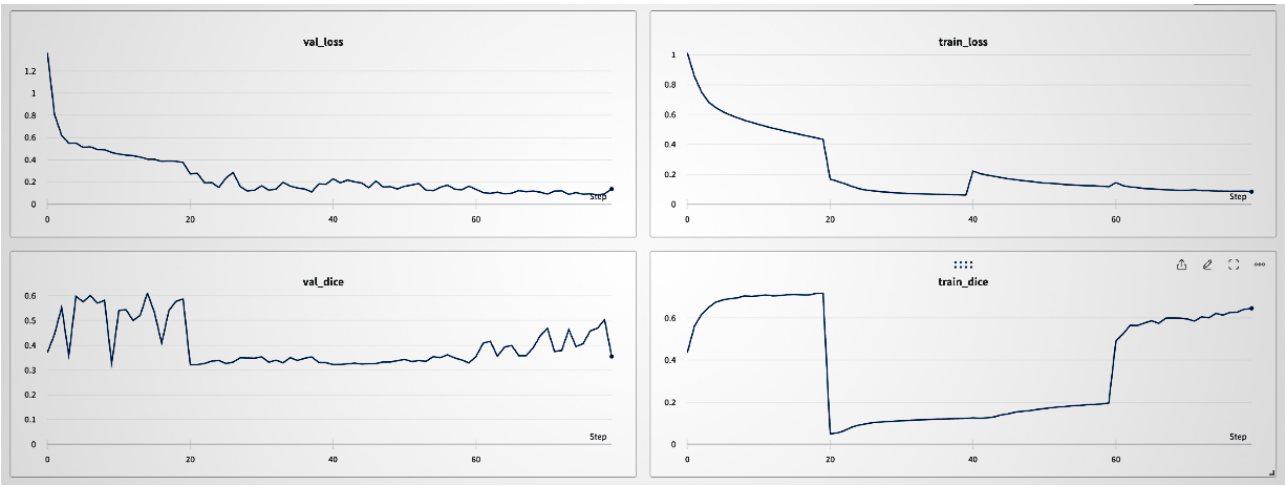


Figure 3.2 Metrics Log when trained with sub-volume batches

In the Dice scores, encompassing Dice0, Dice1, and Dice2, signify the MeshNet model's precision in delineating distinct classes or regions within the dataset. Their counterparts, Val-Dice-0, Val-Dice-1, and Val-Dice-2, reflect the model's validation set performance across these

specific categories. These scores are pivotal indicators of the model's accuracy in segmenting diverse elements within the data. Simultaneously, Train-Loss and Val-Loss metrics play a crucial role by quantifying the disparity between predicted and actual values during the training and validation phases, respectively. Lower values in these metrics denote enhanced model convergence and predictive accuracy.

The Step-LR metric likely tracks learning rate variations throughout training steps, offering insights into the model's adaptive learning behavior. Furthermore, the aggregated validation Dice score (val-dice) provides a comprehensive evaluation of the model's effectiveness in segmenting or classifying data elements on the validation set. Collectively, these metrics offer nuanced insights into the MeshNet model's performance, providing a holistic assessment of its segmentation capabilities and validation set generalization. They serve as fundamental tools for comprehensively evaluating the model's effectiveness and behavior throughout the training and validation phases.

Figure 3.3 Metrics with Onecycle Step LR

CHAPTER 4
METHODOLOGY

4.1 Introduction

In our methodology, we initially utilized Google Collab's minimum GPU specifications for training our MeshNet model. We then expanded this methodology to our decentralized framework with the goal of minimizing architectural memory usage and simplifying the training process. This approach was chosen to maximize computational efficiency and optimize resource utilization during the training phase.

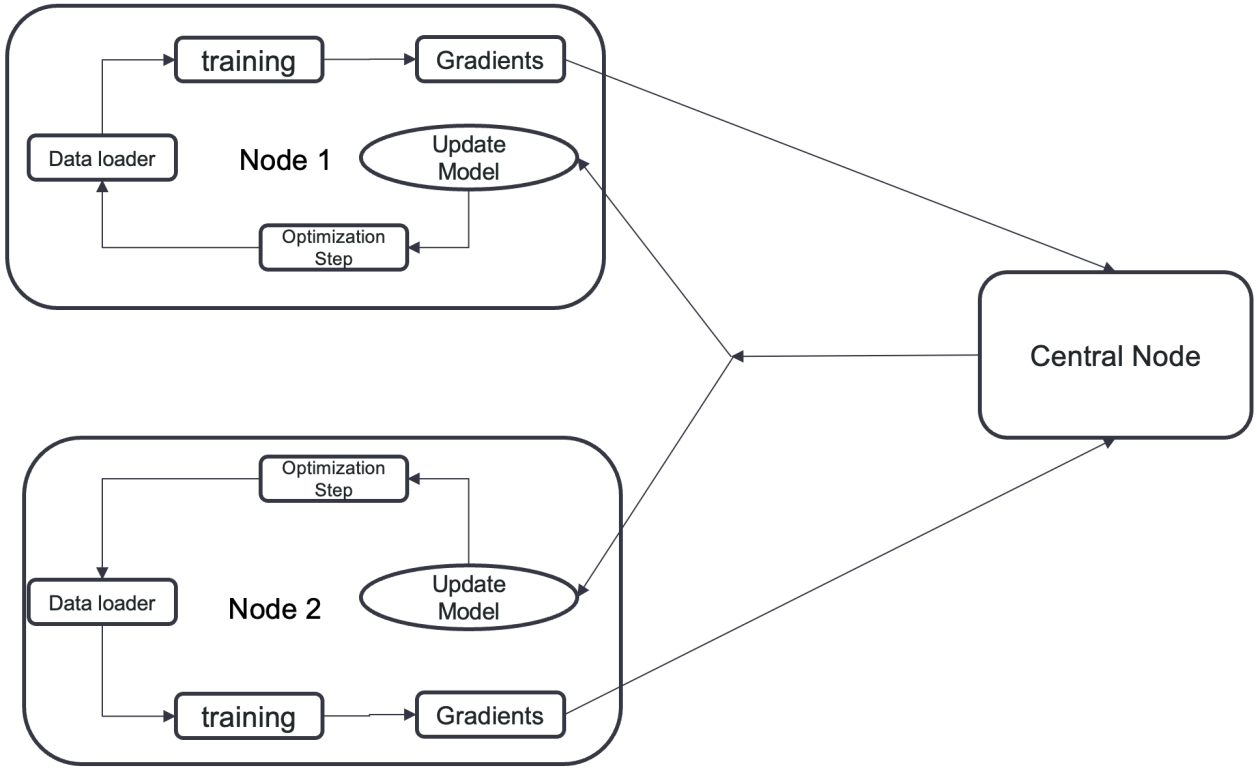


Figure 4.1 Coinstac Decentralized Architecture

4.2 Distributed learning using Gradient aggregation

Centralized Gradient Aggregation is a strategy employed in distributed learning to streamline the process of updating machine learning models across multiple nodes or machines. In this method, instead of each node independently updating its model based on local data, all nodes send their model updates or gradients to a central server or point. At this central location, these updates are aggregated or combined to create a unified view of the model's performance across the entire network. This aggregated view provides valuable insights into the overall direction and effectiveness of the learning process.

Listing 4.1 Central Node Algorithm

```
FUNCTION avg_grads(args):
    INPUT: args - gradients data remote nodes
    OUTPUT: agg_grad - list of averaged gradients
    grads = [input_list[node] for node IN input]
    sum_arrs = [sum(arrays) for arrays IN zip(*grads)]
    avg_arrs = [sum_arr / len(grads) for sum_arr IN sum_arrs]
    RETURN avg_arrs
    END FUNCTION
```

Once the central server has synthesized this collective view, it distributes the information back to each node. This feedback guides individual nodes on how to adjust their models to better align with the overall performance goals. By leveraging this synchronized approach, Centralized Gradient Aggregation promotes collaboration among the nodes, ensuring that

they collectively contribute to improving the model's accuracy and efficiency.

Listing 4.2 Remote Node Algorithm

```
FUNCTION remote_node_algorithm(local_gradients):
    FOR EACH epoch IN range(num_epochs):
      FOR EACH batch IN range(num_batches):
        model.train()
        send_gradients_to_central(local_gradients)
        agg_gradients = receive_agg_gradients()
        update_model_gradients(agg_gradients)
        perform_optimization_step()
        send_acknowledgment_to_central_node()
    END FUNCTION
```

Overall, Centralized Gradient Aggregation facilitates a coordinated and cohesive learning process in distributed settings, where diverse data sources and computational resources are utilized. This method fosters a unified improvement cycle, enabling distributed systems to iterative refine and enhance machine learning models over time.

This thesis presents the implementation and experimentation of two distinct approaches for decentralized learning within a Meshnet deep learning model framework.

4.2.1 Coinstac Application

Using Coinstac (16; 1)for distributed training with the Meshnet model simplifies research collaboration. Users analyze their data on their own computers, keeping their information private. It scales well by spreading the workload across many machines. Collaborating is easier because users contribute without sharing sensitive data. Results are replicable and transparent since each user runs the same analysis. Coinstac offers flexibility in hardware and software choices, reducing data transfer and setup complexities, making distributed computing accessible for researchers.

4.2.1.1 Implementation and workflow

Constack facilitates local testing of simulations using Docker containers. The simulator includes a Docker-file defining container architecture, an input spec file for required inputs, local.py and remote.py for local-remote communication logic, and input folders for each local node. Through Constac, computations are orchestrated seamlessly using Docker containers for local testing.

I have developed a new simulator for training the MeshNet (7) model in a distributed environment. This simulator allows users to input various components necessary for training, including the MeshNet model file in Python, a dice function, an SQLite database file containing compressed image and label data in a table format, and a dataloader for reading data from the database. Additionally, users specify parameters such as epochs, learning rate, classes, and a WandB URL for monitoring training progress.

During computation, the simulator automatically (12) deploys a set of Docker images representing local nodes and one central node. Each training cycle on a local node involves sending gradients to the remote node, which then waits for all nodes to transmit their gradients. Once the remote node receives gradients from all local nodes, it calculates aggregated gradients and sends back the averaged gradients to each local node. This ensures that all nodes are updated with the same aggregated gradients (8), facilitating synchronized training iterations until successful completion.

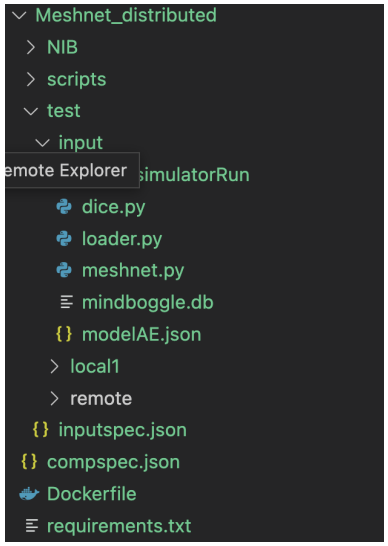


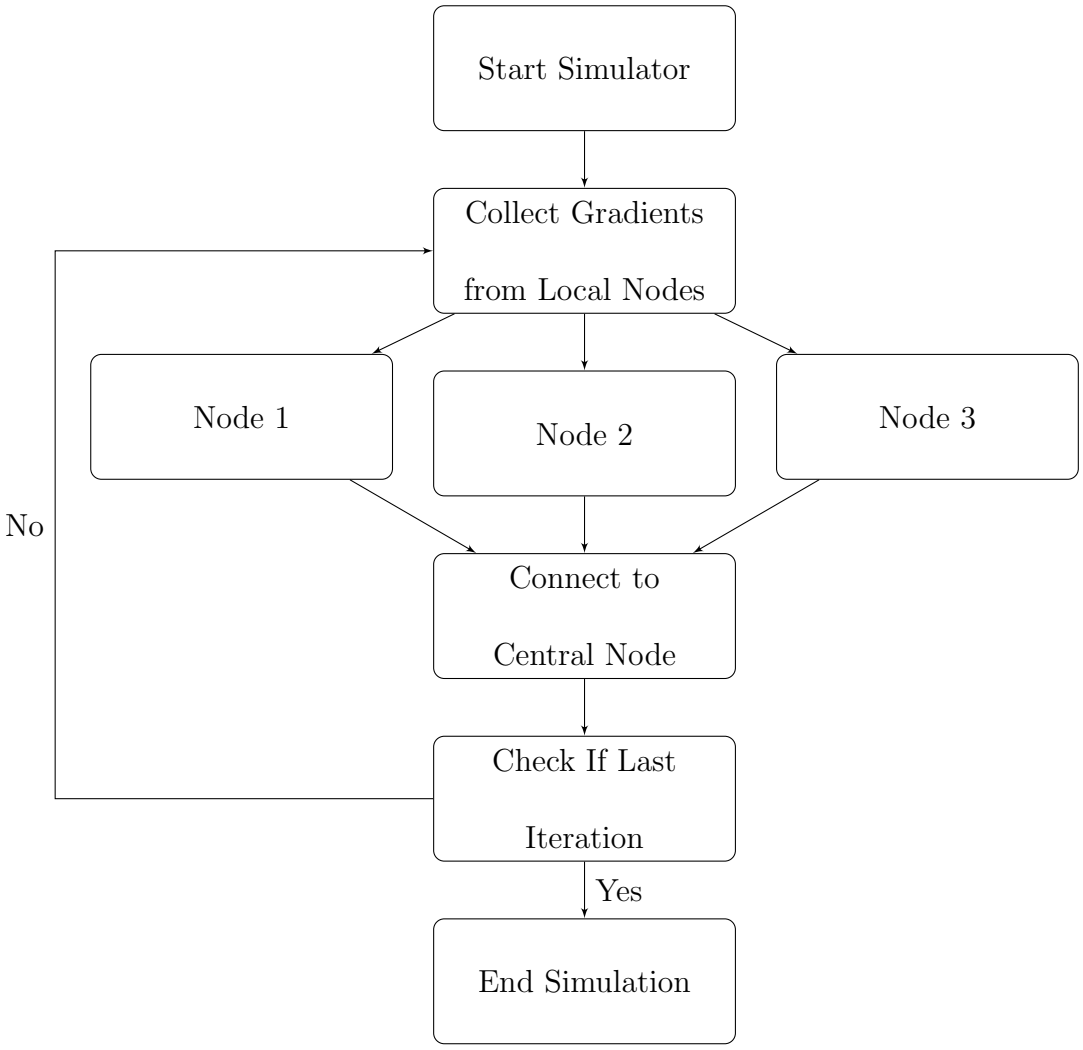
Figure 4.2 Simulator file system structure

To monitor training progress and view logs, the simulator integrates WandB logging, collecting metrics from each node and visualizing them in real-time during training. This comprehensive approach to distributed training enhances collaboration and efficiency, making it easier for researchers to train complex models like MeshNet (7) in a distributed simulation environment.

The simulator can be seamlessly integrated with the Coinstac desktop application. Within

Coinstac (1), users can create a consortium and establish a pipeline (2), featuring computation steps such as Meshnet distributed. Input covariates are provided for data inputs, and additional users can join the pipeline and upload required files. Once the requisite users are assembled, the computation can commence within the consortium pipeline.

Distributed Gradient Aggregation architecture



Local Simulation

As mentioned Coinstac (16) also provided feasibility to test local simulton. To initiate a local simulation using the simulators developed for distributed learning available in the Meshnet_distributed folder of the repository mentioned in (12), follow these steps:

- Ensure to install all the necessary requirements mentioned in (15).
- Download the repository containing the latest simulator from the provided link and navigate to the Meshnet_distributed folder within the repository.
- To begin the local simulation, we need to create a Docker image as instructed in (16). This image will contain the necessary environment and dependencies.
- After creating the Docker image, execute the coinstac-simulator command. This command will deploy a one remote node container specific number of local node containers based on the inputs specified in the inputspec file which can be visble in docker desktop.

Desktop Application simulation

- Open the Coinstac desktop application and locate the Consortium feature within the in the main menu.
- Choose to either create a new consortium for your simulation or join an existing one. Define the consortium's parameters like its name and purpose.
- Establish a pipeline for your simulation within the consortium. Outline the computation steps needed for the simulation process in our case it is **MeshNet-distributed**

- Define the specific computation steps within each stage of the pipeline. Specify parameters and dependencies as required.
- Input the necessary covariates or variables required for the simulation. This may include data inputs, parameters, or initial conditions.
- Once the pipeline is set up and data provided, start the computation process within the consortium. Monitor its progress and review the results once the simulation is complete.

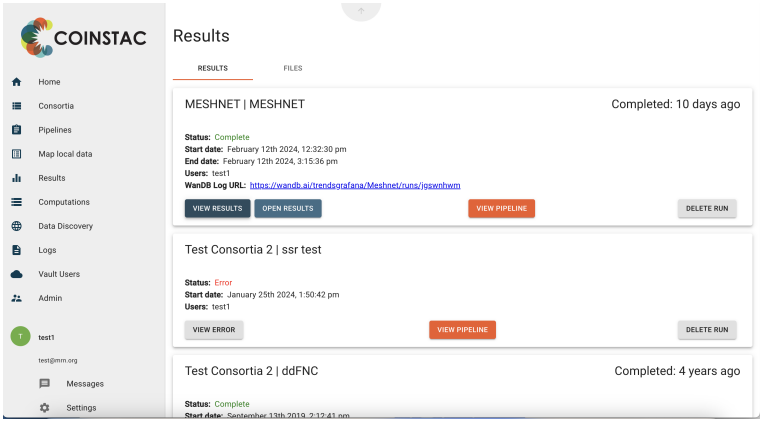


Figure 4.3 Simulation Reference

4.2.2 Serverless Gradient Aggregation Framework

The implementation phase of this research project focused on developing a robust and efficient system for aggregating gradients in a distributed computing environment. Leveraging cloud-native services provided by Amazon Web Services (AWS), the implementation aimed to integrate various components seamlessly while ensuring security, scalability, and reliability throughout the system architecture.

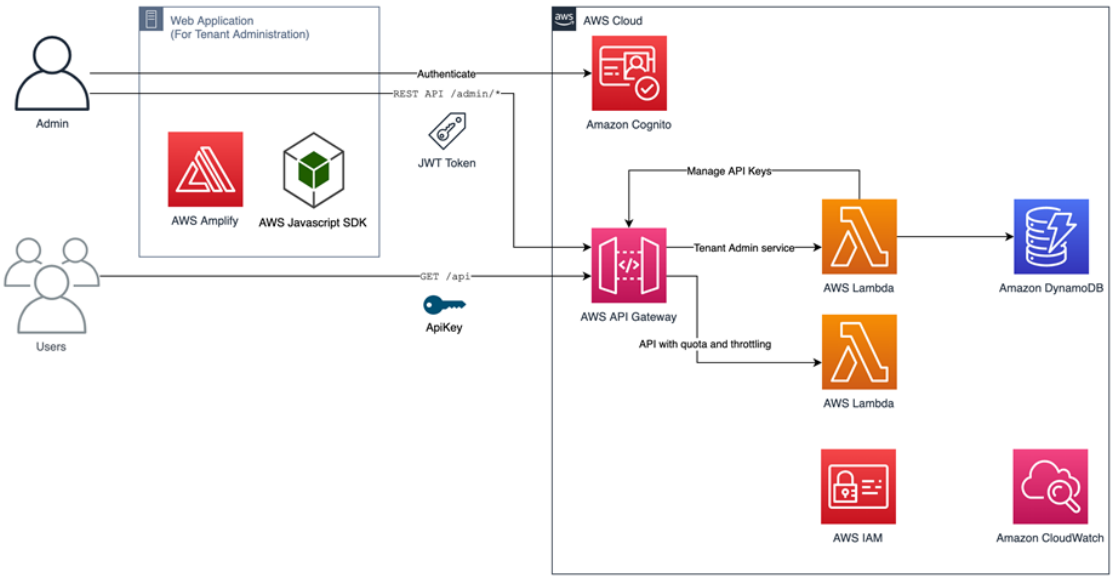


Figure 4.4 AWS Architecture Diagram - Distributed gradient aggregation

AWS Cognito was utilized as the primary mechanism for user authentication and identity management. By integrating Cognito with the system, users could securely authenticate themselves using a variety of authentication methods supported by the service. This ensured that only authorized users could access the system's resources, thereby safeguarding sensitive data and maintaining the integrity of the system.

API Gateway played a pivotal role in managing the RESTful APIs exposed to users. By integrating API Gateway with Cognito, the system enforced strict authentication and authorization policies for accessing the APIs. This integration allowed for seamless user authentication while offloading the complexity of managing user identities and access control to AWS services, streamlining the development process and enhancing the overall security posture of the system.

The server-less computing paradigm offered by AWS Lambda was leveraged to implement

the backed logic for processing user requests and aggregating gradients. Lambda functions were designed to scale automatically in response to fluctuating workload demands, eliminating the need for manual provisioning and optimizing resource utilization. By encapsulating the computation logic within Lambda functions, the system achieved greater flexibility and agility in handling user requests while minimizing operational overhead.

DynamoDB served as the underlying database for storing user activities and relevant system data. By restricting access to DynamoDB to only authorized Lambda functions, the system enforced fine-grained access control and ensured that data access was limited to trusted components within the system. This approach enhanced data security and integrity while providing a scalable and cost-effective storage solution for the system.

Overall, the implementation phase successfully integrated AWS services to develop a resilient and scalable system for aggregating gradients in a distributed computing environment. By leveraging cloud-native technologies and best practices, the system achieved its objectives of providing secure, efficient, and reliable gradient aggregation capabilities while laying the foundation for future enhancements and optimizations.

Below figure shows an workflow of communication between each service.

4.2.2.1 Implementation and workflow

The implementation of our project relies heavily on the utilization of Terraform, an industry-leading Infrastructure as Code (IaC) tool provided by HashiCorp (11). Terraform offers a

streamlined approach to deploying and managing infrastructure, enabling us to define our desired architecture in code and provision it across various cloud environments.

Deployment of back-end architecture into AWS account :

By leveraging Terraform for our project implementation, we gain several benefits. Firstly, with Infrastructure as Code (IaC), our infrastructure is defined and managed as code, facilitating version control, collaboration, and repeatability. Secondly, Terraform's automation capabilities streamline the provisioning and configuration of infrastructure, reducing manual effort and minimizing the risk of human error. Additionally, Terraform enables scalability, allowing us to easily adjust our infrastructure to accommodate changing demands by modifying our Terraform configuration. Finally, Terraform ensures consistency across environments, enabling reliable replication of our infrastructure across various stages, including development, testing, and production environments. Overall, these features empower us to efficiently manage and scale our infrastructure while maintaining consistency and reliability throughout the project lifecycle.

The deployment process using Terraform involves the following steps:

- **Configuration**: We define our infrastructure requirements in Terraform configuration files, specifying the desired state of our architecture.

- **Initialization**: We initialize Terraform in our project directory using the terraform init command. This step downloads the necessary providers and modules required for our infrastructure.

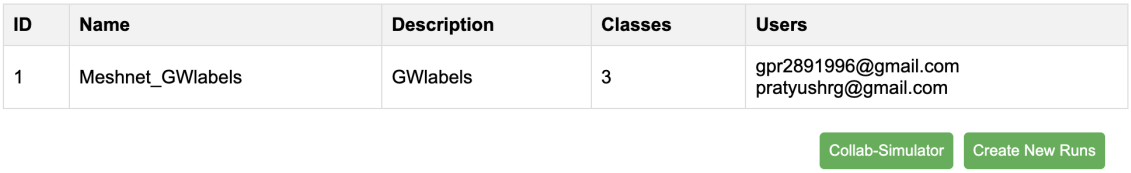


Figure 4.5 Coinstac simulator management console

- **Planning**: We generate an execution plan using terraform plan, which provides a preview of the actions Terraform will take to create, update, or delete resources based on our configuration.

- **Execution**: We apply the Terraform configuration using terraform apply, which triggers the provisioning of our infrastructure based on the execution plan generated in the previous step.

- **Validation**: We verify that the deployed infrastructure meets our requirements and functions as expected.

Upon successful deployment of the simulator we can be able to login to the simulator web application to create and manage simulations for meshnet

Simulator for Restful frame work :

I have developed a flask application to utilize the the rest frame work for train model in any system architecture with minimum installations required. Our Flask application serves as a comprehensive platform for managing and executing machine learning simulations (14) in a distributed setting. The workflow begins with user authentication, where individuals access the application through a secure login process. Upon successful authentication, users

gain access to the application's functionalities, primarily centered around the initiation and monitoring of machine learning simulations. Through the user interface, individuals can start new simulations, review ongoing tasks, and access historical data, streamlining the management of complex experiments.

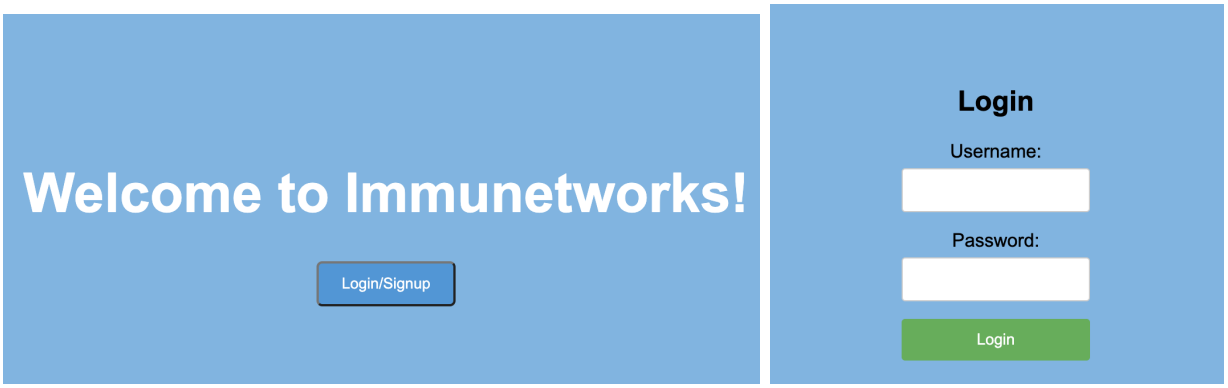


Figure 4.6 Authentication using rest framework

Once a user initiates a new simulation, the Flask application orchestrates the training process using a multi-threaded approach. This enables concurrent execution of training tasks, optimizing resource utilization and expediting the training process. Each thread within the multi-threaded environment independently handles various aspects of the training pipeline, including model initialization, data loading, training loop execution, and gradient exchange with the distributed system. This decentralized approach ensures efficient distribution of computational workload and enhances the scalability of our application. Throughout the training process, the Flask application facilitates seamless communication with the distributed system. Gradients computed during training are exchanged between the application and distributed nodes through API calls. Leveraging these exchanges, our application enables collaborative learning across multiple nodes, enhancing the robustness and effectiveness

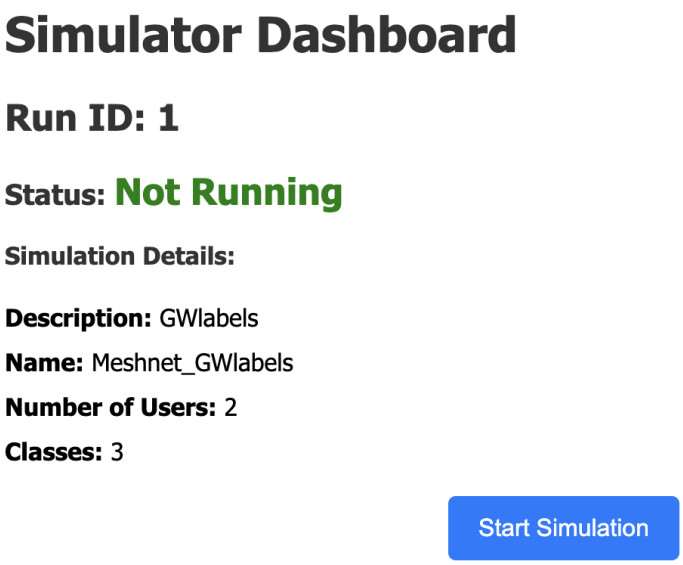


Figure 4.7 Simulator dashboard

of the machine learning models being trained. Additionally, the application's main thread remains responsive, allowing users to monitor simulation progress in real-time and providing timely feedback on the status of their experiments.

Periodic validation checks are conducted to ensure the accuracy and reliability of the training process. These checks involve evaluating model performance metrics against validation data, verifying convergence, and identifying potential issues or anomalies. Upon completion of training, simulations are deactivated, and users gain access to comprehensive results and insights through the web interface. Simulation results, including performance metrics, training logs, and visualizations, empower users to analyze experiment outcomes effectively and derive meaningful conclusions from their research endeavors. Overall, our Flask application offers a sophisticated yet user-friendly platform for conducting distributed machine learning experiments, facilitating collaboration, and advancing research in the field.

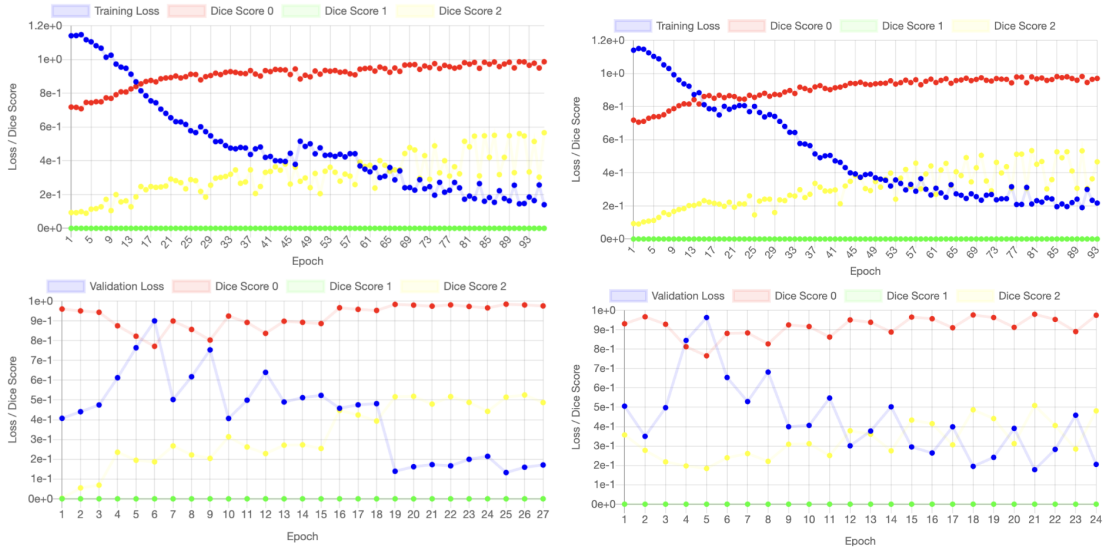


Figure 4.9 Sample logged Metrics in different nodes

CHAPTER 5
RESULTS

In the results section of our thesis documentation, we present the outcomes of our simulation testing, which involved training the meshnet model from its initial stage without utilizing any pre-trained weights. This training was conducted in two distinct distributed simulators: one implemented within the Coinstac application, utilizing remote nodes, and the other a custom Restful simulator employing the AWS Lambda serverless approach for gradient aggregation calculation.

Through rigorous comparison, we evaluated the performance of the models trained in both scenarios. Specifically, we assessed models trained for the same number of iterations and using identical datasets. This comparative analysis provided valuable insights into the efficacy and efficiency of the workflow in each setting.

5.1 Utilizing Coinstac Distributed Simulator

In this section, we present the results obtained from training the Meshnet model using the Coinstac distributed simulator. The training dataset comprised 10 batches of training data, 3 batches of validation data, and 2 batches of test data. Each data point in the dataset consists of serialized information containing images and labels, both of shape $256\hat{3}$.

During the training process, we adopted a methodology where each volume in the dataset was partitioned into subvolumes of varying shapes. Specifically, we employed subvolume shapes of 32^3, 128^3, 64^3, and 256^3 across separate training sessions, each spanning 10 epochs.

In Dice scores, encompassing Dice0, Dice1, and Dice2, signify the MeshNet model's

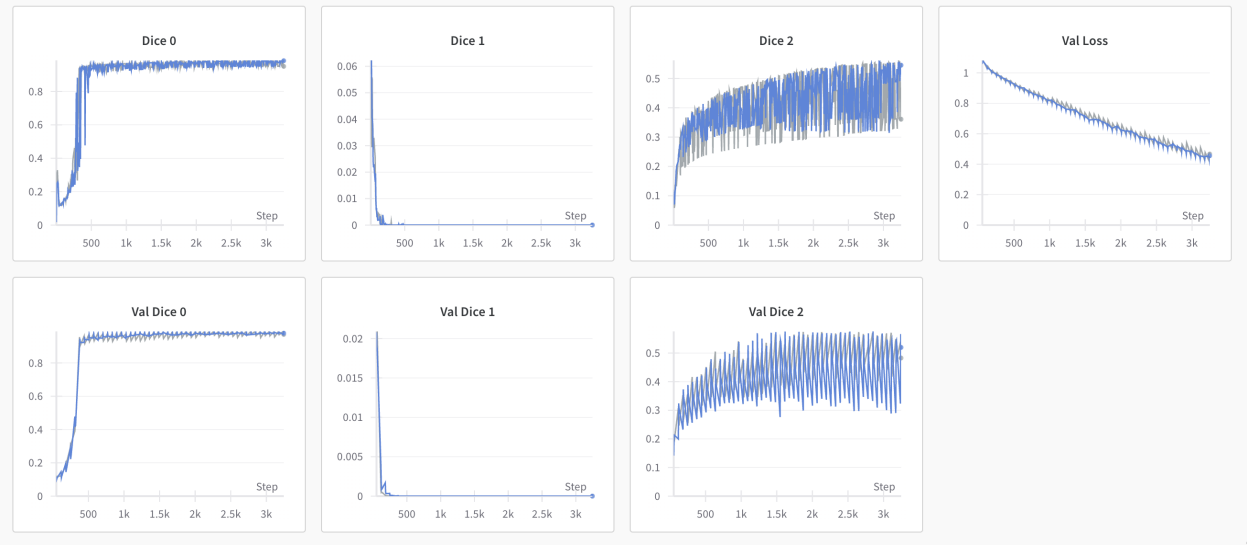


Figure 5.1 Centralized Gradient Aggregation Logging from Coinstac

precision in delineating distinct classes or regions within the dataset. Their counterparts, ValDice0, ValDice1, and ValDice2, reflect the model's validation set performance across these specific categories. These scores are pivotal indicators of the model's accuracy in segmenting diverse elements within the data.

Simultaneously, TrainLoss and ValLoss metrics play a crucial role by quantifying the disparity between predicted and actual values during the training and validation phases, respectively. Lower values in these metrics denote enhanced model convergence and predictive accuracy. Furthermore, the aggregated validation Dice score (valdice) provides a comprehensive evaluation of the model's effectiveness in segmenting.

Collectively, these metrics offer nuanced insights into the MeshNet model's performance, providing a holistic assessment of its segmentation capabilities and validation set generalization. They serve as fundamental tools for comprehensively evaluating the model's ef-

fectiveness and behavior throughout the training and validation phases. The decentralized approach ensures fair task distribution among nodes, fostering balanced contributions.

5.2 Utilizing Immunetworks Distributed Simulator

The results obtained for Immnetworks distributed system are quite promising. Achieving good accuracy with a dice score of 80% indicates strong performance in the task at hand. This level of accuracy demonstrates the effectiveness of the system in processing and analyzing data across distributed nodes.

Training on two different nodes over the full volume data of 15 datasets, with each dataset containing 10 test and 3 validation datasets, showcases the system's ability to handle large-scale data efficiently, Figure 5.2. The fact that these results were achieved after training for 80 epochs on each node further emphasizes the robustness and reliability of the distributed training approach.

Overall, these results highlight the effectiveness of Immnetworks in distributed systems, showcasing its potential to handle complex tasks with high accuracy and efficiency. This bodes well for its applicability in various domains requiring distributed data processing and analysis.

The comparison of predicted and original labels Figure 5.3 using our best-trained model provides valuable insights into its performance. These visualizations aid in understanding the model's strengths and weaknesses, guiding future refinements and optimizations for enhanced accuracy and reliability.

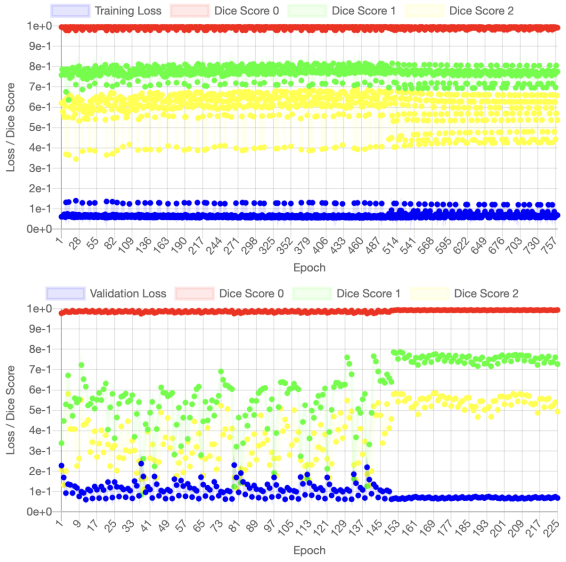


Figure 5.2 Immunetworks Training log

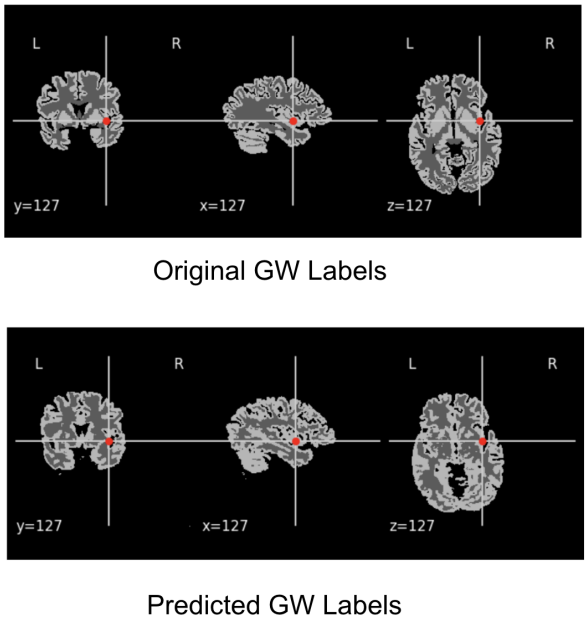


Figure 5.3 Plots for original vs predicted brain scan labels

CHAPTER 6
Discussions and Conclusion

Our strides in decentralized MeshNet learning demonstrate promising advancements in balanced node training. By partnering with Coinstac, we have enhanced our approach for impactful distributed learning strategies. Additionally, the integration of Immunetworks, a lightweight RESTful framework, has streamlined our system's scalability and ease of deployment, further empowering our distributed learning architecture.

6.1 Contributions

- **Enhanced Distributed Learning Strategies :** Through collaboration with Coinstac, we have developed and optimized distributed learning strategies that leverage local data while ensuring efficient communication and coordination among nodes. This has resulted in more effective utilization of computational resources and improved model performance.

- **Development of Immunetworks :** The incorporation of Immunetworks, a lightweight RESTful framework, has simplified the deployment and scalability of our distributed learning system. This contribution addresses the practical challenges associated with deploying complex distributed systems in real-world settings.

6.2 Implications

- Our work has significant implications for the future of collaborative learning. By decentralizing the training process and integrating efficient communication protocols,

we pave the way for scalable and collaborative approaches to model training across diverse datasets and computational environments.

- The techniques and frameworks developed in this project offer solutions to the challenges of training deep learning models on distributed data sources. These advancements have implications for industries such as healthcare, finance, and manufacturing, where large-scale data analysis is essential, but resources are often distributed across different locations.

- The success of our project highlights the potential for broader adoption of distributed learning techniques in various domains. As the demand for analyzing and extracting insights from massive datasets continues to grow, decentralized approaches to machine learning become increasingly relevant and necessary.

6.3 Future Directions:

- As our project continues to evolve, ensuring the scalability and robustness of our distributed learning system will be paramount. This includes addressing challenges related to handling increasingly large and diverse datasets, as well as improving fault tolerance and resilience to node failures.

- There is potential for exploring alternative architectures and models beyond MeshNet for distributed learning tasks. Investigating the suitability of different neural network architectures and their performance in decentralized environments could lead to further advancements in the field.

- Future efforts could focus on applying our distributed learning framework to real-world problems and deploying it in practical settings. This may involve collaborating with industry partners to address specific challenges and validate the effectiveness of our approach in real-world scenarios.

Through this ongoing project, we have identified potential advancements in collaborative learning, aiming to enhance efficiency and scalability. In conclusion, our collaborative efforts in decentralized MeshNet learning, supported by Coinstac and utilizing Immunetworks, have made significant progress in balanced node training. This project has the potential to contribute to the evolution of distributed learning, providing more efficient and scalable solutions.

REFERENCES

[1] Coinstac. https://github.com/trendscenter/coinstac-computation, 2023.

[2] Coinstac simulator. In GitHub repository. GitHub. send-data-across-local-remote. https://github.com/trendscenter/coinstac-computation2-send-data-across-local—remote-example, 2023.

[3] NIfTI reader. (2017). GitHub.https://github.com/rii-mango/ NIFTI-Reader-JS4, 2023.

[4] Alex Fedorov, Eswar Damaraju, Vince Calhoun, Sergey Plis . Almost instant brain atlas segmentation for large-scale studie. https://arxiv.org/abs/1711.00457 , 2017.

[5] Alex Fedorov, Jeremy Johnson, Eswar Damaraju, Alexei Ozerin, Vince Calhoun, Sergey Plis . End-to-end learning of brain tissue segmentation from imperfect labeling. https://arxiv.org/abs/1612.00940 , 2017.

[6] Brett Matthew1, Markiewicz Christopher, Hanke, Michael, Côté, Marc-Alexandre. nipy/nibabel: 5.1.0. https://zenodo.org/records/7795644, 2023.

[7] David C. Van Essen, Stephen M. Smith, Deanna M. Barch, Timothy E.J. Behren, Essa Yacoub, Kamil Ugurbil. The WU-Minn Human Connectome Project: An overview. https://doi.org/10.1016/j.neuroimage.2013.05.041, 2013.

[8] Fisher Yu, Vladlen Koltun . Multi-scale context aggregation by dilated convolutions. https://arxiv.org/abs/1511.07122 , 2016.

[9] Mohamed Masoud 1, Farfalla Hu 1, and Sergey Plis. Brainchop: In-browser MRI volumetric segmentation and rendering. doi:10.21105/joss.05098 , 2023.

[10] Mohamed Masoud, Pratyush Reddy, Farfalla Hu, Sergey Plis . Brainchop: Next Generation Web-Based Neuroimaging Application. https://arxiv.org/abs/2310.16162 , 2023.

[11] HashiCorp. Terraform. https://www.terraform.io/, 2024.

[12] Pratyush G. Coinstac-meshnet. https://github.com/trendscenter/coinstac-meshnet.git, 2024.

[13] Pratyush G. immunetworks-terraform. https://github.com/neuroneural/immunetworkstf.git, 2024.

[14] Pratyush G. immunetworks-tsimulator. https://github.com/PratyushGR/immunetworksapp.git, 2024.

[15] Trends Center. Coinstac-first-example. https://github.com/trendscenter/coinstac-first-example.git, 2023.

[16] Trends Center. Coinstac-trends. https://github.com/trendscenter/coinstac.git, 2023.