Comparative Analysis of Centralized, Federated, and Swarm Learning Approaches for Image Classification on the CIFAR-10 Dataset

Sai Prasad Komuroju, Purnima Sai Koumudi Panguluri

skomuroju1@student.gsu.edu, ppanguluri1@student.gsu.edu

Abstract— **Advancements in the field of machine learning have unveiled diverse methodologies for tackling image classification tasks, offering a spectrum of strengths and potential applications. This study conducts a comparative investigation into three distinct machine learning paradigms as they are applied to the CIFAR-10 dataset—a collection representative of the challenges inherent in real-world image classification. The traditional strength of Centralized Learning is benchmarked against the data privacy-conscious approach of Federated Learning (FL), which distributes computations across multiple nodes. In contrast, Distributed Swarm Learning (DSL) takes inspiration from the collective intelligence found in nature, such as flocks of birds or schools of fish, to foster a collaborative and decentralized problem-solving environment. The practical execution of DSL methods through simulation highlights the balance between computational efficiency and accuracy, providing insights into their suitability for scenarios where data privacy and system robustness are of paramount concern. By exploring the nuances of these approaches, this paper endeavors to provide a clear understanding of the trade-offs involved in selecting an appropriate model for image classification challenges, particularly under constraints that necessitate a judicious allocation of resources.**

I. Introduction

Machine learning has rapidly become a driving force behind many innovations, transforming industries by making sense of large volumes of data and automating tasks that once required human intelligence. Amongst the diverse spectrum of machine learning methodologies, Convolutional Neural Networks (CNNs), Federated Learning (FL), and Distributed Swarm Learning (DSL) have distinguished themselves, each embodying unique attributes and offering distinct solutions to the enigma of image classification.

Convolutional Neural Networks, the paragon of visual recognition, have cemented their status through their deployment across a myriad of real-world applications. From the nuanced intricacies of facial recognition in security systems to the meticulous analysis required in medical diagnostics, and extending to the cutting-edge domain of autonomous vehicular navigation, CNNs have consistently showcased their capability to dissect and interpret complex visual data with astounding accuracy.

Federated Learning emerges as a beacon of privacy in the data-centric world of machine learning. By facilitating the construction of a collective, robust model while preserving the sanctity of individual data, FL addresses the growing concerns over privacy and data sovereignty. Its applications are manifold and significant, from enhancing the predictive capabilities of smartphone keyboards without compromising user privacy to accelerating the pace of collaborative pharmaceutical research, FL is paving the way for a new era of privacy-conscious machine learning.

Distributed Swarm Learning, with its roots in the natural world, harnesses the emergent intelligence of biological collectives. Emulating the decentralized decision-making prowess of avian flocks or ichthyic schools, DSL is at the vanguard of algorithms tailored for distributed problem-solving in the absence of centralized governance. This paradigm finds resonance in real-world scenarios such as decentralized sensor networks, where a harmonious collective approach is crucial for navigating through dynamic environments and in the presence of fragmented information.

The genesis and evolution of these methodologies are emblematic of the continuous pursuit for more intelligent, secure, and autonomous systems capable of surpassing human limitations. The CIFAR-10 dataset, an established benchmark within the machine learning community, provides a fertile ground for the comparative exploration of these diverse methodologies. Our investigation aims to illuminate their inherent strengths and limitations, fostering an understanding that will inform their deployment in various real-world contexts.

The subsequent discourse will delve into the intricacies of the Centralized Learning model underpinned by CNNs, the Federated Learning model that simulates a collaborative yet decentralized network, and the Distributed Swarm Learning model, which weaves together the threads of artificial and biological intelligence. This exposition aims to unravel the conceptual frameworks and practical applications of each

model, elucidating their potential to address the challenges that lie at the frontier of machine learning.

II. Related Work

The exploration of machine learning models for image classification has been extensive, with significant contributions that have established Convolutional Neural Networks (CNNs), Federated Learning (FL), and Distributed Swarm Learning (DSL) as powerful tools in the AI toolkit.

A. Convolutional Neural Networks

CNNs have been a foundational technology in the realm of image processing and computer vision. Originating in the late 1980s with neural networks capable of recognizing basic patterns in images, the advent of CNNs marked a transformative shift. Their architecture, inspired by the human visual cortex, is specifically designed to process pixel data and recognize patterns that make up an image, such as edges, shapes, and textures. Over the years, improvements in hardware and algorithms have allowed CNNs to handle increasingly complex image recognition tasks, revolutionizing industries by enabling applications such as automated medical diagnosis, security surveillance with real-time facial recognition, and more.

B. Federated Learning

Federated Learning (FL) emerged from the need to handle data in a privacy-preserving manner, facilitating the training of machine learning models without the need to centralize sensitive data. This is particularly important in scenarios where data privacy is paramount, such as in personalized medicine or individualized recommendations based on user data. FL allows multiple participants or devices to collaboratively learn a shared prediction model while keeping all the training data on the device, decoupling the ability to do machine learning from the need to store the data in the cloud.

C. Distributed Swarm Learning

Distributed Swarm Learning (DSL) is an approach inspired by biological processes, specifically the behavior of swarms such as flocks of birds, schools of fish, or colonies of ants. These systems leverage the decentralized decision-making properties of swarms, which can be particularly effective in scenarios where a single central controller isn't feasible or effective. In machine learning, DSL algorithms apply these principles to optimize network parameters in a manner that mimics this collective intelligence, potentially leading to more robust and scalable models that can operate in dynamic or challenging environments.

The aforementioned phrase establishes the framework for delving into the ways in which these technologies have enhanced the field of machine learning while also pushing the frontiers of artificial intelligence, specifically with regard to picture classification. Every approach has a distinct set of advantages and tackles a different set of problems, demonstrating the versatility and diversity of contemporary AI technologies. The study seeks to provide additional insights into the performance and practical uses of these models by putting them to use and testing them on a standard dataset such as CIFAR-10. This will help to advance the ongoing evolution of machine learning methodologies.

III. Experiment

3.1 Convolutional Neural Network

In our experiment with the CIFAR-10 dataset, we employed a sequential Convolutional Neural Network (CNN) comprising multiple convolutional, pooling, and dense layers, combined with dropout for regularization and batch normalization to improve convergence.

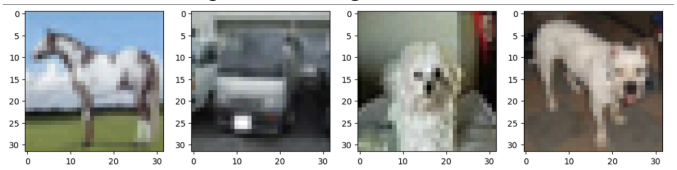

Fig 1: Image samples of CIFAR 10 Dataset

Below, we provide a detailed breakdown of the architecture and the functionality of each component:

3.1.1 Convolutional Layers

Layer Details: The first layer is a 2D convolutional layer with 32 filters, a kernel size of 3×3 'same' padding to maintain the spatial dimensions, ReLU activation function, and 'glorot_normal' initializer.

$$\mathbf{A}_{ij}^{(l)} = f\left(\sum_{m,n} \mathbf{K}_{mn}^{(l)} \cdot \mathbf{A}_{i+m,j+n}^{(l-1)} + b^{(l)}\right)$$

where $A^{(l)}$ is the activation at layer l, $K^{(l)}$ is the kernel, $b^{(l)}$ is the bias, and f is the ReLU activation function.

3.1.2 Batch Normalization

Applied after each convolutional layer, it normalizes the activations of the previous layer, enhancing the training speed and stability.

$$\hat{x}_i = \frac{x_i - \mu_{\mathcal{B}}}{\sqrt{\sigma_{\mathcal{B}}^2 + \epsilon}}$$

where $\mu_{\mathcal{B}}$ and $\sigma_{\mathcal{B}}^2$ are the mean and variance of the batch $\mathcal{B}$, and ϵ is a small constant to avoid division by zero.

3.1.3 Max Pooling

Reduce spatial dimensions by taking the maximum value over 2×2 window, applied after certain convolutional layers.

3.1.4 Dropout

Applied after pooling layers and dense layers with a rate of 0.3, it randomly sets input units to 0 at each step during training time, which helps prevent overfitting.

3.1.5 Flatten Layer

Converts the 3D feature maps into 1D feature vectors necessary for the fully connected layers.

3.1.6 Dense Layers

The network includes dense layers with 512 and 128 units, respectively, both using ReLU activation. The final dense layer has 10 units with a softmax activation function suitable for multi-class classification.

Training

Optimizer: Adam (Adaptive Moment Estimation) is a popular optimization algorithm in deep learning because it combines the best properties of the AdaGrad and RMSProp algorithms to handle sparse gradients on noisy problems. It's an extension to stochastic gradient descent that has recently seen broader adoption for deep learning applications in computer vision, natural language processing, and other computational fields.

Key Features of Adam:

Adaptive Learning Rate: Adam calculates an adaptive learning rate for each parameter. It maintains a separate learning rate for each network weight (parameter) which is adapted as learning unfolds.

Moment Estimation: Adam also keeps an exponentially decaying average of past gradients (similar to momentum), which helps to accelerate the optimizer in the right direction, thus leading to faster converging.

Bias Correction: It implements bias corrections to the first and second moments, which counteract the biases towards zero at the start of training, thereby improving stability.

Update rule for weight *w* at time step *t*:

$$w_{t+1} = w_t - \frac{\eta \cdot \hat{m}_t}{\sqrt{\hat{v}_t} + \epsilon}$$

where, η is the learning rate, $\widehat{m}_t$ is the bias-corrected first moment estimate, $\widehat{v}_t$ is the bias-corrected second moment estimate and ϵ is a small scalar used to prevent division by zero.

Loss Function - Categorical Crossentropy: Categorical crossentropy is a loss function that is used in multi-class classification tasks. These are tasks where each input sample should belong to one of K classes. The function measures the performance of a classification model whose output is a probability value between 0 and 1.

$$L = -\sum_{i=1}^{N}\sum_{k=1}^{K} y_{i,k}\log(\hat{y}_{i,k})$$

where, N is the number of samples, K is the number of classes, $\mathcal{Y}_{i,k}$ is a binary indicator (0 or 1) if class label k is the correct classification for sample i and $\widehat{\mathcal{Y}}_{i,k}$ is the predicted probability that sample i, belongs to class k.

Procedure

Epochs: The model is trained for 50 epochs, which means the learning algorithm will work through the entire dataset 50 times. This number of epochs is chosen to allow sufficient time for the model to converge on a solution.

Batch Size: A batch size of 32 is used, meaning 32 samples from the dataset are used to estimate the error gradient before updating model weights. This size is a balance between training speed and model performance.

Data Normalization: Input data is normalized to have a mean of 0 and a standard deviation of 1. This helps in speeding up the learning process and reaching a faster convergence.

Validation Data: Used to monitor the model's performance on unseen data. It helps in identifying issues like overfitting and helps in tuning the hyperparameters.

3.2 Federated Learning (FL) Model

In our experiment with Federated Learning (FL) setup, we utilized a simulated environment where multiple agents (or clients) collaboratively train a model without sharing their local data. This setup is particularly useful in scenarios where data privacy is crucial or when data cannot be centralized due to bandwidth or regulatory constraints.

3.2.1 Data Distribution

Each agent receives a portion of the training data. The dataset is divided equally among the agents, ensuring each subset is used to train local models independently.

3.2.2 Local Model Training

The model comprises around 10 Epochs for each and every individual agent. There are around 5 agents defined. Every agent trains a local model using their subset of the data. This training is performed locally without exchanging raw data with the central server or other agents. The dataset is split into chunks for each client to simulate a distributed data environment. A new local model is created for each client, initialized with the weights of the global model. Each client's model is trained on its local dataset. The client's model weights are averaged with the global model weights to simulate a federated averaging step. This represents the FL aggregation mechanism.

3.2.3 Aggregation

After training, each agent sends their model updates (typically weights) to the central server. The server aggregates these updates to update the global model. Common aggregation techniques include Federated Averaging

$$\theta_{global} = \frac{1}{N}\sum_{k=1}^{N}\theta_k$$

where N is the number of agents and θ_k are the parameters from the k^{th} agent.

3.2.4 Model Architecture

The model is a Convolutional Neural Network (CNN) suitable for image classification tasks. Convo2D Layers are convolutional layers that will extract features from the images. They have 64 and 128 filters respectively. Max Pooling layers reduce the spatial dimensions (width and height) of the output volume. The flattened layer flattens the input and is used as a bridge between the convolutional layers and the dense layers. Fully connected layers that provide learning capabilities to the network. The final dense layer has 10 units corresponding to the 10 classes in the CIFAR-10 dataset.

The updated global model is then evaluated to measure its performance, often using a held-out validation set that was not seen during training.

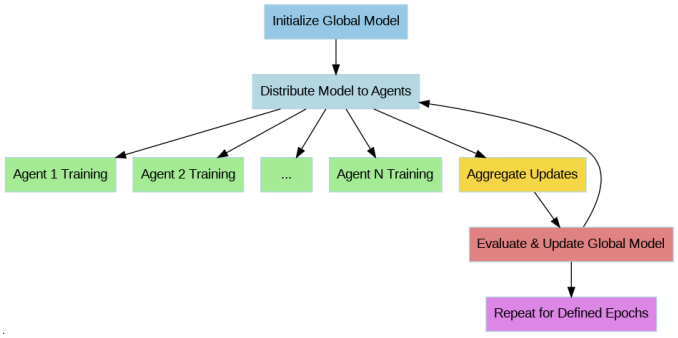

Fig 2: Conceptual Flowchart of Federated Learning

3.2.5 Iteration

The process is repeated for several rounds (epochs), with the global model being refined each time.

3.3 Distributed Swarm Learning (DSL) Model

The Distributed Swarm Learning (DSL) model we're implementing integrates concepts from swarm intelligence with deep learning. Here, each agent (or node) independently trains a model using a subset of data, while employing Particle Swarm Optimization (PSO) principles to collaboratively improve a global model. This approach not only maintains data locality and privacy but also benefits from collective intelligence found in swarm behavior.

3.3.1 CNN Model Architecture

The CNN is the underlying architecture for each agent in our swarm system. Each agent uses a Convolutional Neural Network (CNN) designed for image classification tasks. The network includes layers typical for handling image data: convolutional layers for feature extraction, pooling layers to reduce spatial dimensions, dropout layers to prevent overfitting, and dense layers for classification.

3.3.2 Particle Swarm Optimization (PSO)

In DSL, PSO is adapted to optimize the neural network weights across different agents. Each agent's model weights are treated as particles in the swarm, where each particle adjusts its position (weights) based on its own experience (personal best) and that of its neighbors (global best).

3.3.3 PSO Mechanics

Velocity Update: Each agent's model weights are updated by adjusting the velocity vector based on the following formula

$$v_{i,t+1} = w \cdot v_{i,t} + c_1 \cdot r_1 \cdot (pbest_i - x_{i,t}) + c_2 \cdot r_2 \cdot (gbest - x_{i,t})$$

where, $v_{i,t}$ is the current velocity of agent i and t, w is the inertia weight, controlling the impact of the previous velocity on the current one, c_1 *and* c_2 cognitive and social coefficients, r_1 *and* r_2 are random numbers between 0 and 1, p_{best} is the personal best position of the particle, g_{best} is the global best position of the swarm among all agents, $x_{i,t}$ is the current position (weights) of agents i and t.

Position Update: The new position (weights) of each particle (agent's model) is updated by:

$$x_{i,t+1} = x_{i,t} + v_{i,t+1}$$

3.3.4 Flow of the DSL Model Training

The flowchart for DSL model training can be visualized as follows:

Step 1: Initialization

Local Models: The number of agents defined in the model is 5. Each agent starts with an individual copy of the neural network model, which could either be initialized randomly or based on a pre-defined starting point.

PSO Parameters: Important PSO parameters like inertia weight (w), cognitive coefficient (c1), and social coefficient (c2) are set. These parameters dictate how much agents rely on their own best performance versus the swarm's best-known performance.

Step 2: Data Distribution

Each agent receives a subset of the overall dataset, ensuring that the data is both diversified and sufficient to train a robust model. This setup protects data privacy since no raw data is shared among the agents or with a central server.

Step 3: Local Training

Each agent trains their model locally using their specific data subset:

Training Process: Agents use standard training techniques, adjusting weights based on backpropagation from the loss calculated on their local data.

Local Epochs: Training can consist of several epochs locally before any synchronization or updating with the global model.

Step 4: Update Velocity and Position
After a round of local training, each agent updates their model's weights (position in the solution space) using PSO rules:
Velocity Update: Each agent calculates a new velocity based on their own historical best performance (personal best) and the global best performance seen across the swarm.
Position Adjustment: The new velocity influences the weights directly, nudging the model towards a potentially better region in the solution space.

Step 5: Evaluate Global Model
Post-update, agents evaluate their models to determine if there have been improvements:
Local Validation: Agents test their models against a local validation set to assess performance improvements.
Global Communication: Each agent communicates their performance metrics to a coordinator or directly to other agents.

Step 6: Update Global Best
If an agent's model outperforms the current global best:
Global Model Update: The agent's model weights are adopted as the new global best.
Broadcast: The new global best is shared with all agents to serve as a reference point for further velocity and position updates.

Step 7: Adjust PSO Parameters
Based on the outcomes and dynamics observed:
Adaptation: Parameters such as inertia weight, and cognitive and social coefficients might be adapted to improve convergence rates or exploration/exploitation balance.

Step 8: Repeat for Epochs
The process repeats for several epochs:
Our model takes 25 epochs as a whole. Each epoch involves local training, updating of velocities and positions, evaluation, and potential updating of the global model. The loop continues until predetermined stopping criteria are met (such as a maximum number of epochs or a satisfactory error threshold). After the Final epoch, the global model is evaluated on the test dataset to assess its overall performance and generalization capability. Metrics such as loss and accuracy are reported, providing insight into the efficacy of the collective learning process.

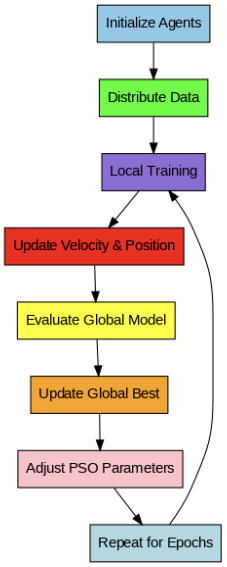

Fig 3: Flow of the DSL Model Training

IV. Results

4.1 Performance Evaluation of CNN
Training Accuracy: The model has achieved a high training accuracy of approximately 94%. This indicates that the model has a strong capacity to learn from the dataset and can fit the training data well.
Validation Accuracy: The validation accuracy hovers around 86%, which is slightly lower than the training accuracy. This gap suggests that while the model performs well on unseen data, there might be a slight overfitting to the training data, as it performs better there.
Test Accuracy: The test accuracy also stands at approximately 85.66%. This is consistent with the validation accuracy, affirming the model's ability to generalize to new data, which is essential for practical applications.
Loss: The loss on both the training and validation sets is showing a decreasing trend, which is a positive sign of learning. The lower validation loss compared to the training loss in the last few epochs could be due to the regularization effects of dropout in the model.

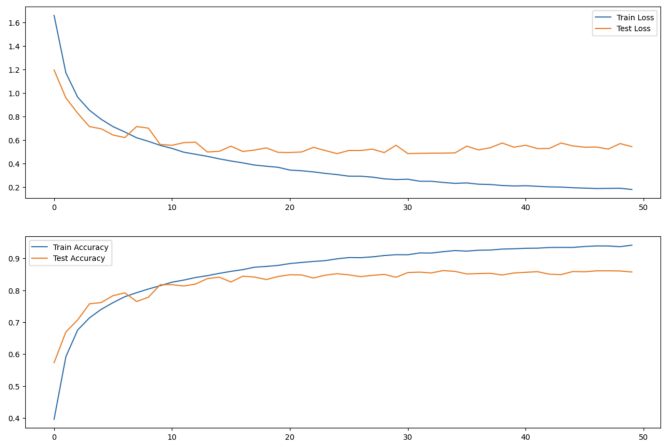

Fig 4: Train and Test Accuracy and Losses Comparisons

Confusion Matrix: Each row of the matrix represents the instances of an actual class, while each column represents the instances of a predicted class. The diagonal cells (from the top left to the bottom right) represent the number of points for which the predicted label is equal to the true label (correct predictions), while the off-diagonal cells represent the misclassified points (incorrect predictions).

The classes with the highest number of correct predictions are classes 1 (automobiles) and 9 (trucks), which could suggest that the model is particularly good at distinguishing these categories.

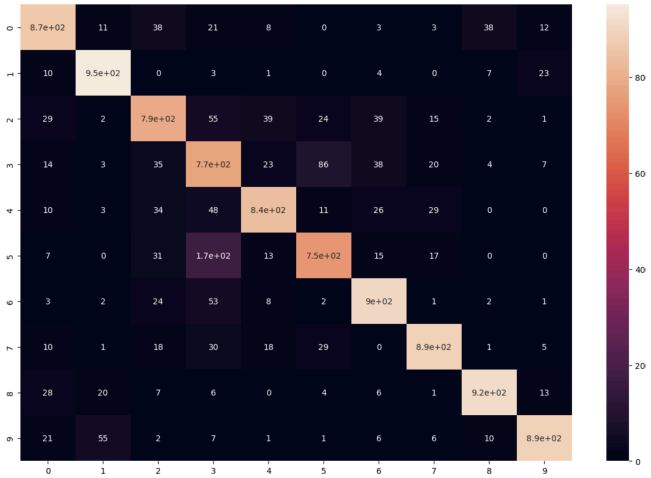

Fig 5: Confusion Matrix

Computational Efficiency

Epoch Duration: The training time per epoch is relatively long, taking several minutes. As the model complexity increases with additional layers and parameters, the computational cost goes up, which is visible from the time per step during training epochs.

Batch Processing: The model processes each batch in approximately 288-340ms in the final epochs of training, which is typical for models of this complexity when trained on standard hardware.

4.2 Performance Evaluation of Federated Learning

Agent Performance Variation: Each agent seems to have a varying degree of accuracy, with the lowest being approximately 70.91% and the highest around 71.77%. Each agent has different slices of data, which may contain various distributions and complexities.

Training and Test Accuracy: The training and testing accuracies show that the model is learning and is able to generalize to an extent. The test accuracy has been recorded at approximately around 70.92%.

Loss Metrics: The loss values appear to decrease over epochs, which is expected in a well-functioning training process. A discrepancy between training loss and validation loss can be an indicator of the model's ability to generalize from training data to unseen data.

Consistency: Across agents, if the performance metrics (accuracy and loss) are consistent, it suggests that the Federated Learning approach is working well. Significant discrepancies might require further investigation into data distribution or agent model configurations.

4.3 Performance Evaluation of Distributed Swarm

Performance Metrics:

Test Loss: This metric indicates how well the model is doing in terms of making predictions on the test set. A lower test loss value is better as it signifies that the model's predicted outputs are closer to the true values.

Test Accuracy: This represents the percentage of correct predictions made by the model out of all predictions. Higher accuracy means the model is performing better at classifying the test images into one of the 10 classes of CIFAR-10. The test accuracy of the model is recorded to be approximately 61.07%.

Evaluation Over Epochs:

The distributed swarm learning model starts with a relatively high loss and low accuracy, as is typical at the beginning of the training process. As the epochs progress, we observe a general trend of decreasing test loss and increasing test accuracy, indicating that the model is learning and improving its predictive capabilities.

There are epochs where the test loss slightly increases or the test accuracy plateaus or decreases, which is normal in a training process due to various factors such as model's exploration of the solution space, learning rate adjustments, or the inherent stochastic nature of the learning algorithm.

The final epoch (Epoch 25) shows the best test accuracy and the lowest test loss, suggesting that the model has converged to a reasonably good solution.

Performance Evaluation:

The gradual decrease in test loss combined with the increase in test accuracy indicates a positive learning trajectory. The model's ability to improve over time suggests that the distributed swarm approach is effectively utilizing the collective knowledge of individual agents to enhance the global model.

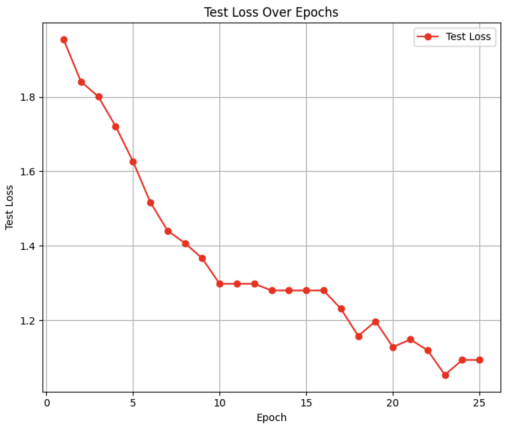

Fig 6: Test loss over Epochs

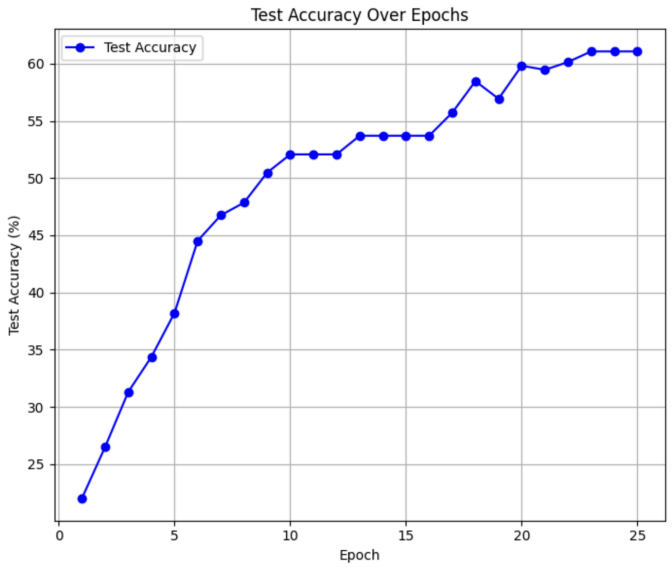

Fig 7: Test accuracy over Epochs

V. Model Execution Manual

This section provides instructions on how to execute Convolutional Neural Networks (CNNs), Federated Learning (FL), and Distributed Swarm Learning (DSL) models for image classification tasks on the CIFAR-10 dataset. The code can be run in Google Colab or a Jupyter Notebook environment.

5.1 Prerequisites

- A Google account for accessing Google Colab.
- For local execution, Python 3.x installed on your machine.
- An Integrated Development Environment (IDE) like Jupyter Notebook.
- Required Python libraries: tensorflow, numpy, matplotlib, sklearn, and graphviz.

5.2 Setup Instructions for Jupyter Notebook

1. Install Jupyter (if not already installed):
   - Open a terminal or command prompt.
   - Run `***pip install notebook***`
2. Launch Jupyter Notebook:
   - In the terminal, run `***jupyter notebook***`
   - This will open Jupyter in your default web browser.

5.3 Installation of Required Libraries

In a new cell in your notebook, type and run the following commands to install the necessary libraries:

```
!pip install tensorflow numpy matplotlib sklearn graphviz
```

5.4 Downloading the Dataset

The CIFAR-10 dataset can be automatically loaded via TensorFlow's dataset library. Use the following command to load the dataset:

Note: This step is already included in the code. (For external usage)

```
from tensorflow.keras.datasets import cifar10
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
```

5.5 Running the Model

Implement the model provided or open the *.ipynb* file given. Compile the model with the given optimizer, loss function, and metrics.

5.6 Saving the Model

To save the trained model to your workspace, use:

Note: This step is already included in the code. (For external usage)

```
model.save('my_model.h5')
```

5.7 Testing with our own dataset

In a new cell in your notebook, you can read your test file using the appropriate method for its format. For instance, if it's a h5 file: (Google Colab)

```
# Load the model
model = load_model('/content/cnn.h5')  # Replace with the actual path to your model file
```

and finally evaluate the model using the following command:

```
# Evaluate the model on the test dataset
loss, accuracy = model.evaluate(test_images, test_labels, verbose=1)
print(f"Test Loss: {loss}")
print(f"Test Accuracy: {accuracy}")
```

By following these instructions, you should be able to upload your test data successfully and proceed with evaluating your machine learning models.

VI. Conclusion

Through the implementation and evaluation of three distinct machine learning models—Convolutional Neural Networks (CNNs), Federated Learning (FL), and Distributed Swarm Learning (DSL)—on the CIFAR-10 dataset, this study has highlighted the strengths and weaknesses of each approach in terms of accuracy, generalization to unseen data, and computational efficiency.

Model Performance Summary

The CNN model showcased strong performance with high training accuracy and good generalization as evidenced by its validation and test accuracies.

The FL model, designed to train collaboratively while preserving data privacy, demonstrated moderate performance with some variability among agents.

The DSL model incorporated swarm intelligence principles, resulting in a unique collaborative training approach. Despite achieving lower accuracy compared to the CNN model, it offered insights into alternative learning paradigms that could be particularly useful in decentralized environments.

Implications and Limitations

The implications of these findings are significant for applications requiring data privacy and distributed data sources. While the federated and swarm learning approaches need further refinement to match the accuracy of centralized methods, they offer promising directions for scenarios where data cannot be shared. A notable limitation of this study is the fixed number of epochs and agents. Further experiments with a larger number of agents and extended training could provide more insight into the scalability and dynamics of FL and DSL models.

References

[1] "ImageNet Classification with Deep Convolutional Neural Networks" by Alex Krizhevsky, Ilya Sutskever, and Geoffrey E. Hinton.

[2] "Deep Learning for Visual Understanding: A Review" by A. Alom, T. M. Taha, C. Yakopcic, S. Westberg, P. Sidike, M. Nasrin, M. Hasan, and V. K. Asari.

[3] "Communication-Efficient Learning of Deep Networks from Decentralized Data" by H. Brendan McMahan, Eider Moore, Daniel Ramage, Seth Hampson, and Blaise Aguera y Arcas.

[4] "Advances and Open Problems in Federated Learning" by Peter Kairouz, H. Brendan McMahan, Brendan Avent, Aurélien Bellet, and Mehdi Bennis.

[5] "A Review of Swarm Intelligence Algorithms" by Daniel T. Curiel-Ramirez, Oscar Montiel-Ross, and Elisa Schaeffer.

[6] "Distributed Swarm Intelligence for Multirobot Systems: A Review" by Gabriella A. B. Barros, Matheus M. Silveira, and Marley M. B. R. Vellasco.