Feature Align-HFL : A Feature Alignment Based Client-Specific Model For Heterogeneous Federated Learning

Thesis Submitted for Partial Fulfillment
of the Requirements for the Award of the Degree of

Master of Technology

in

Electronics & Communication Engineering

Specialization: Communication and Signal Processing

by

Jyotirmoy Nath
Roll Number : 2302010

Under the supervision of

Dr. Shovan Barma

Department of Electronics & Communication Engineering

Indian Institute of Information Technology Guwahati

12 th November, 2024

CERTIFICATE FOR APPROVAL

This is to certify that the thesis entitled **"Feature Align-HFL: A Feature Alignment Based Client-Specific Model For Heterogeneous Federated Learning"** submitted by Jyotirmoy Nath to Indian Institute of Information Technology Guwahati, is a record of bona fide research work under my supervision and I consider it worthy of consideration for the award of the degree of Master of Technology in Electronics and Communication Engineering at Indian Institute of Information Technology Guwahati.

Dr. Shovan Barma

Associate Professor

Department of Electronics and Communication

Indian Institute of Information Technology, Guwahati

Guwahati 781001, Assam, India.

DECLARATION

I declare that

1. The work contained in the thesis is original and has been done by myself under the general supervision of my supervisor.

2. The work has not been submitted to any other Institute for any degree or diploma.

3. I have followed the guidelines provided by the Institute in writing the thesis.

4. I have conformed to the norms and guidelines given in the Ethical Code of Conduct of the Institute.

5. Whenever I have used materials (data, theoretical analysis, and text) from other sources, I have given due credit to them by citing them in the text of the thesis and giving their details in the references.

6. Whenever I have quoted written materials from other sources, I have put them under quotation marks and given due credit to the sources by citing them and giving the required details in the references.

Jyotirmoy Nath

Abstract

—Federated Learning (FL) enables decentralized model training while ensuring data privacy between clients and the server, and is applicable in various domains such as healthcare, finance, and edge computing. Traditional FL ap proaches assume homogeneous model architectures across clients and servers, limiting their applicability in real-world scenarios where clients use diverse models. To address this, we introduce Feature Align-HFL, a novel framework designed for Heteroge neous Federated Learning (HFL) that aligns dissimilar models for effective knowledge sharing. Specifically, we demonstrate its application by aligning CNNs and U-Nets for common tasks. By employing cosine similarity , Feature Align HFL identifies and aligns common features across heterogeneous models. Experiments show that CNN and adapted U-Net models achieved feature map similarities reaching up to 96 percent on a common task. These results highlight the significant transfer able knowledge between dissimilar architectures, indicating that Feature Align-HFL can effectively support knowledge sharing in heterogeneous FL settings.

Keywords: Heterogeneous Federated Learning (HFL), Feature Alignment, Client-Specific Models

Acknowledgement

I would like to express my deepest gratitude to my Supervisor, **Dr. Shovan Barma**, Associate Professor, Department of Electronics and Communication, Indian Institute of Information Technology Guwahati, for his valuable guidance, consistent encouragement, personal care, and timely help, as well as for providing an excellent environment for conducting my research. Despite his busy schedule, he extended cheerful and cordial support, without which I would not have been able to complete this project work.

I also extend my thanks to all the teaching and non-teaching staff of the Electronics and Communication Department, Indian Institute of Information Technlogy Guwahati, for their constant support and encouragement. .

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Background of Federated Learning

Federated Learning (FL) is an innovative approach to machine learning in which data remains decentralized, allowing model training across multiple devices or servers without centralizing the data itself. Introduced by Google in 2016, FL enables a collaborative model training process by leveraging data stored locally on devices like smartphones, IoT devices, or distributed servers. Each device trains a local model on its data, and only the model updates (not the raw data) are shared with a central server, which aggregates these updates to improve the global model. This decentralized training framework addresses challenges of data privacy, security, and data accessibility that are increasingly critical in the digital age.

1.1.1 Advantages of Federated Learning

FL offers several distinct advantages over traditional machine learning (ML), especially in contexts where data privacy, efficiency, and scalability are critical. Here are some key advantages of Federated Learning :

1. **ENHANCED PRIVACY AND SECURITY**
    - Raw data remains secure on local devices and is never transmitted to central servers

- Complies naturally with data protection regulations like GDPR and CCPA
- Reduces the risk of massive data breaches since data is distributed

2. **REDUCED NETWORK BANDWIDTH**
   - Only model updates and parameters are transmitted across the network
   - Significantly reduces data transfer costs compared to centralized learning
   - Enables efficient learning even with limited or intermittent connectivity
3. **IMPROVED PERSONALIZATION**
   - Models can adapt to individual user preferences while maintaining privacy
   - Enables location-specific and device-specific optimizations
   - Balances personalization with global model improvements

1.1.2 End-to-End Federated Learning Process

Federated learning begins with the central server establishing and distributing a global model to edge devices. Each device performs local training on its private data, keeping sensitive information secure while optimizing model performance locally. Devices then send compressed, encrypted model updates back to the central server, which aggregates them using algorithms like Federated Averaging (FedAvg) and filters out any erroneous contributions. The updated global model is redistributed to all devices, synchronizing the latest version across the network. This iterative process continues, with the system monitoring convergence and adjusting parameters, until the model meets target performance metrics while upholding privacy and efficiency.

Fig. 1 depicts a federated learning system where, a central server coordinates training across multiple clients. Each client has its private data and unique local model architecture labeled as *Model A* for Client 1 and *Model k* for the kth client. Each client performs local updates F_kW on its data and sends these updates to the central server. The central server aggregates these updates, weighted by the number of samples n_k on each client, to form a global model W.

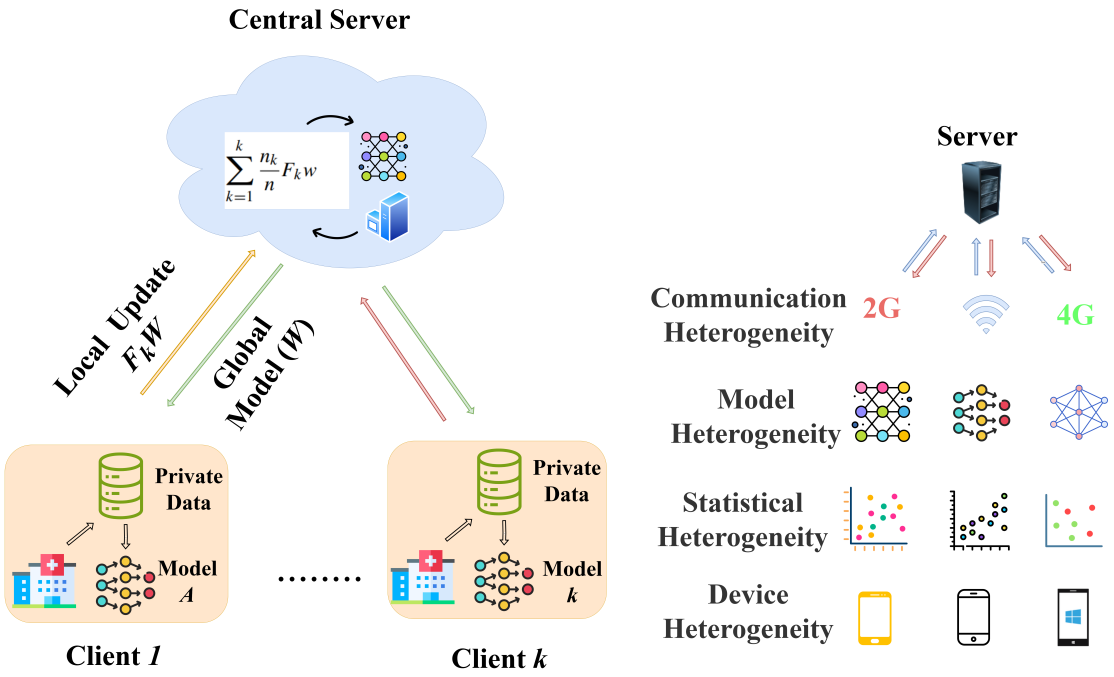

FIGURE 1.1: Federated Learning with Heterogeneous Clients. This diagram depicts a federated learning system where a central server aggregates model updates from diverse clients. Each client features distinct model architectures, statistical data distributions, communication modes (e.g. , 2G, 4G, Wi-Fi), and device types. The illustration captures the complexities and challenges of handling heterogeneity in real-world FL environments.

1.1.3 Types of Heterogeneity in Federated Learning

Federated learning systems face various forms of heterogeneity that significantly impact their performance and implementation.However, FL faces four key heterogeneities as shown in Fig.1.

1. *Statistical Heterogeneity:* Clients often possess non-iid (non-Independent and Identically Distributed) data, leading to inconsistent optimization directions and biased global models. Approaches like FedPer [13] and FedProx [11] address these challenges by personalizing models to account for data distribution differences.

2. *Model Heterogeneity:* Clients may use different model architectures suited to their specific tasks or resources. Consequently, each client may develop its local model independently, which introduces challenges in transferring knowledge across these diverse model structures.

3. *Communication Heterogeneity:* Variability in network bandwidth and connection stability can cause unsynchronized updates, impacting global model convergence. Techniques like FedAvg [6] with asynchronous updates and adaptive strategies [9] mitigate these issues by adjusting update frequencies.

4. *Device Heterogeneity:* Differences in device capabilities, such as computing power and memory, lead to performance disparities among clients. Methods like FedBuff [5] and FedAdapt [9] address these disparities by dynamically adjusting client workloads.

1.1.4 Applications of Federated Learning

Healthcare Applications

- Medical image analysis across different hospitals
- Patient diagnosis prediction using distributed health records
- Drug discovery using data from multiple research centers

Mobile and Edge Computing

- Next word prediction in mobile keyboards
- Face detection in smartphone cameras
- Voice recognition systems on personal devices

Autonomous Vehicles

- Traffic pattern recognition
- Driver behavior analysis
- Vehicle-to-vehicle communication learning

Smart Retail

- Inventory management across stores

- Customer preference prediction
- Personalized shopping recommendations

1.2 Motivation

Federated Learning (FL) has emerged as a promising approach for privacy-preserving distributed model training, but its traditional assumption of identical model architectures across clients limits its real-world applicability. While some solutions attempt to address model heterogeneity, they fall short in handling fundamentally different architectures. We identify the following key challenges that motivate our research:

1. **In real-world scenarios, clients often require different model architectures that are:**
    - Optimized for specific tasks (e.g., CNN for classification, U-Net for segmentation),
    - Adapted to varying resource constraints,
    - Tailored to different client requirements.
2. **Architectural heterogeneity poses major challenges:**
    - Difficulty in knowledge sharing between different architectures,
    - Complications in model collaboration across the federated learning (FL) system,
    - Barriers to effective feature transfer between diverse models.
3. **Current solutions are limited:**
    - Existing approaches only handle minor architectural variations,
    - Most methods focus on differences within the same model type,
    - Few solutions address fundamentally different architectures.

    These limitations result in Restricted potential of collaborative learning.

1.3 Objective

This project aims to :

1. Design and implement a novel framework (Feature Align-HFL) that enables effective collaboration between heterogeneous model architectures in federated learning settings
2. Develop a feature alignment mechanism that can identify and map corresponding features between fundamentally different model architectures (e.g., CNN and U-Net)
3. Create a method using cosine similarity to measure and align feature representations across heterogeneous models

1.4 Contributions

The main contributions of this work are summarized as follows:

1. Introduced Feature Align-HFL, a framework to address
2. Developed a feature alignment method for clients with different architectures.
3. Applied cosine similarity and correlation matrices for effective communication between heterogeneous models.
4. Conducted experiments to validate the framework's ef- fectiveness.

1.5 Organization of the Thesis

The rest of the thesis is organized as follows:

- In **Chapter 2**, we present a comprehensive literature survey of Federated Learning, focusing on heterogeneous federated learning approaches.

- In **Chapter 3**, we detail the methodology of our proposed Feature Align-HFL framework. This includes our problem formulation and system architecture.
- In **Chapter 4**, we outline our experimental methodology, describing the dataset preparation, model architectures (CNN and U-Net), and specific training procedures.
- In **Chapter 5**, we present our results and provide a detailed discussion of our findings.

Chapter 2

Literature Review

Federated Learning (FL), introduced by the Google [2],revolutionizes decentralized model training by enabling collaborative learning across edge devices without centralizing data.Traditional FL methods, such as FedAvg [1], assume model homogeneity across clients, where a central server coordinates the training process and aggregates updates from clients with identical model architectures. This approach preserves privacy by only sharing model parameters but limits flexibility to scenarios where clients employ the same model structure.

Li et al.[11] extend FL to handle non-iid data by incorporat- ing local model regularization to account for data distribution discrepancies. Ningxin Su et al.[10] introduces asynchronous updates to accommodate variations in communication conditions among clients. While these methods enhance practicality, they still operate under the assumption of model homogeneity, which does not address the complexities of heterogeneous model architectures. Knowledge Distillation, initially proposed by Hinton et al. [12], was designed to compress large neural networks by transferring knowledge from a heavy teacher model to a lighter student model.

The teacher model provides logits, or soft outputs, which the student model uses for train- ing. This technique has been extended to FL contexts to handle model homogeneity. Federated Distillation [14] explored the use of logits from homogeneous models within FL, while Kim et al.[15] introduced a paraphraser-student approach to facilitate latent knowledge transfer between networks. Sem- CKD [15] further advanced this field by applying attention mechanisms to align intermediate knowledge across layers.

FedMD [14] and FCCL [7] attempt to facilitate knowledge transfer between heterogeneous models using distillation tech- niques but often focus on variations variations within a single model type, addressing differences such as the number of layers or depths in CNNs or ResNets. However, these approaches are insufficient for scenarios involving fundamentally different architectures, such as CNNs specialized for classification and U-Nets designed for segmentation.

To address these limitations, we propose Feature Align- HFL, a novel framework designed to facilitate knowledge transfer between heterogeneous models without relying on public datasets. By leveraging cosine similarity and correlation matrices, Feature Align-HFL aligns and transfers commonfeatures between models with fundamentally different architectures, such as CNNs and U-Nets. This approach overcomes the limitations of existing FL methods, enabling effective collaboration and improving performance in diverse federated learning environments.

Chapter 3

Methodology

3.1 Problem Formulation

In this Heterogeneous Federated Learning (HFL) context, the problem is defined as follows:

1. **Participants and Local Models:** We consider k participants, each utilizing a local model θ_z for classification tasks. Each participant $z \in \{1, \ldots, k\}$ has access to a dataset D_z. The dataset D_z for participant z is defined as:

$$D_z = \{(X_z, Y_z) \mid X_z \in \mathbb{R}^{N_z \times D}, Y_z \in \mathbb{R}^{N_z \times Q}\} \tag{3.1}$$

   where, N_z, D, and Q refer to the number of samples, input feature dimensionality , and number of classes respectively.

2. **Model Heterogeneity:** The problem involves heterogeneous models, where the local models θ_i and θ_j from different participants i and j have different architectures. This heterogeneity implies that the models are specialized for different tasks. For instance, one model might be optimized for a classification task, while another is optimized for a segmentation task. The architectural differences between models are represented as:

$$\mathcal{M}_i \neq \mathcal{M}_j \text{ for } i \neq j \tag{3.2}$$

where, $\mathcal{M}_i$ and $\mathcal{M}_j$ denote the architectures of the models from participants i and j, respectively. This specialization can impact how effectively the models perform on tasks outside their primary optimization.

3. **Performance on Common Task:** The performance of the models on the common classification tasks is evaluated based on the metric η. Let η_p and η_q denote the accuracy of models from participants p and q, respectively:

$$\eta_p = \frac{1}{N_p} \sum_{i=1}^{N_p} [\hat{y}_i^p = y_i^p] \tag{3.3}$$

$$\eta_q = \frac{1}{N_q} \sum_{j=1}^{N_q} \left[\hat{y}_j^q = y_j^q\right] \tag{3.4}$$

where $[\hat{y}_i^p = y_i^p]$ and $\left[\hat{y}_j^q = y_j^q\right]$ evaluate to 1 if the predicted label $\hat{y}_i^p$ (or $\hat{y}_j^q$) matches the true label y_i^p (or y_j^q), and 0 otherwise. Here, N_p and N_q is the number of samples used for evaluating the accuracy of the model from participant p and q.

The difference in performance between the models is quantified as:

$$\Delta\eta_{pq} = |\eta_p - \eta_q| \tag{3.5}$$

where $\Delta\eta_{pq}$ represents the absolute difference in accuracy between M_p and M_q . This difference quantifies the impact of model heterogeneity on performance across the common classification tasks.

3.2 An Overview of the Proposed Framework

Feature Align-HFL framework, illustrated in Fig. 2, operates within a HFL environment with multiple clients. In the figure, there are k clients, each with its own local data, such as images of birds, dogs, and horses, and each employing a distinct heterogeneous model $M_1, M_2, \ldots, M_k$ that varies fundamentally from the others.

Clients train these heterogeneous models locally on their respective datasets, resulting in feature representations $F_1(x_1), F_2(x_2), \ldots, F_k(x_k)$. These feature representations are then transmitted to a central server, where a Feature Alignment process is conducted to identify a common feature matrix F_{comm} from these diverse models. This aligned representation forms the basis for constructing a global feature model, F_{glob}, which is subsequently shared back with the clients.

This iterative process enables clients to collaboratively enhance their local models while preserving data privacy, as no raw data is exchanged. The system architecture of Feature Align-HFL consists of four key components: training on private data, feature extraction and dimensionality reduction, feature alignment, and identification of common features: which are discussed in the following.

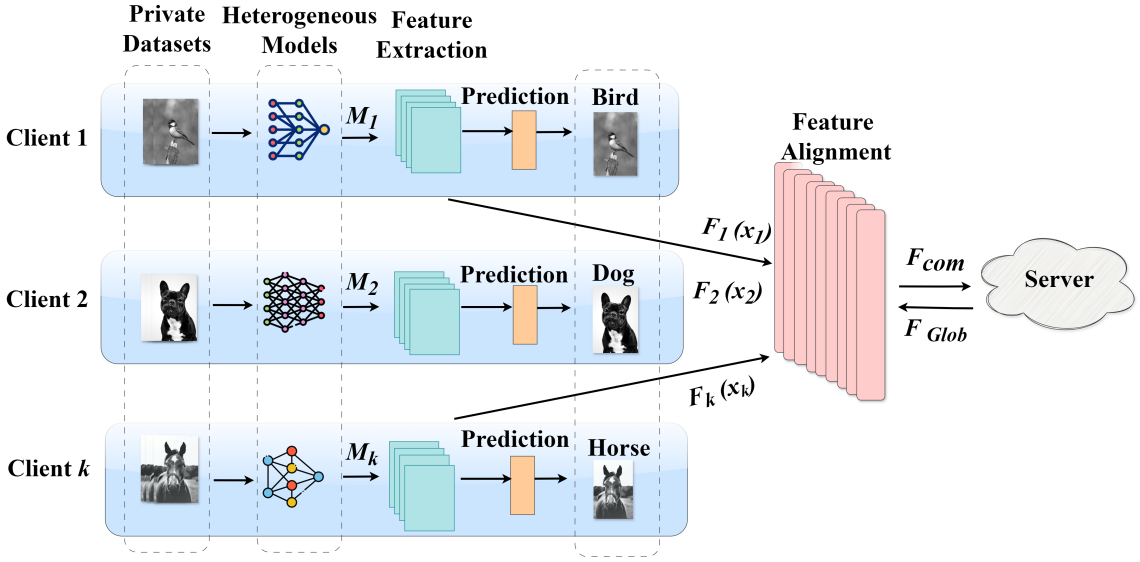

FIGURE 3.1: A system architecture of Feature Align-HFL

1. **Training on Private Data:** Clients initially train their models on their private datasets. This phase involves adapting each model to the unique characteristics of the client's data. The objective function for training the local models is given by:

$$\text{Objective}_l = \min_{M_l} \frac{1}{N_l} \sum_{m=1}^{N_l} L(M_l(w_m, z_m)) \tag{3.6}$$

Here, L is the cross-entropy loss, M_l is the model for client l, and N_l is the number of samples in client l's dataset. w_m and z_m denote the input features

and labels for the m-th sample, respectively. This objective function aims to minimize the average loss across the samples in each client's dataset.

2. **Feature Extraction and Dimensionality Reduction:** After training, clients extract feature representations from their models. To handle high-dimensional feature vectors, each client performs dimensionality reduction using Principal Component Analysis (PCA) to aid in making the dimensions of feature vectors from different models comparable. It ensures that features can be more effectively aligned and compared, enhancing the overall integration process. Let $f_e(x_j)$ represent the feature vector extracted by model M_e for a data point x_j in dataset D_e. The key steps are:

   - **Feature Representation Extraction:** Each client e extracts feature representations $f_e(x_j)$ for its local data.
   - **Dimensionality Reduction Using PCA:** The reduced feature representation $\tilde{f}_e(x_j)$ is obtained as:

$$\tilde{f}_e(x_j) = \text{PCA}(f_e(x_j)) \tag{3.7}$$

3. **Feature Alignment:** Feature alignment identifies and maps corresponding features across different client models, despite architectural differences, to address a common task. For example, CNNs and U-Nets are used for classification and segmentation, respectively.

   To achieve feature alignment, the framework employs one key technique:

   - **Cosine Similarity:** Cosine similarity measures the directional similarity between feature vectors, enabling alignment across models with different architectures. It compares orientations rather than magnitudes, making it suitable for features with varying scales. For reduced feature vectors $\tilde{f}_t(x_j)$ from client t and $\tilde{f}_n(x_l)$ from client n, the cosine similarity is computed as:

$$S_c(\tilde{f}_t(x_j), \tilde{f}_s(x_l)) = \frac{\tilde{f}_t(x_j) \cdot \tilde{f}_s(x_l)}{\|\tilde{f}_t(x_j)\| \|\tilde{f}_s(x_l)\|} \tag{3.8}$$

   where, $\|\cdot\|$ denotes the L2 norm (Euclidean norm) of a vector.

4. **Identification and Aggregation of Common Features:** Common features shared across clients are identified and aggregated by the central server, enhancing the global model's performance. This is done by comparing cosine similarities and correlation matrix entries between clients a and b:

$$F_{\text{com}} = \Big\{ f \in \tilde{f}_a, \tilde{f}_b : S_c(\tilde{f}_a(f), \tilde{f}_b(f)) > \tau \} \tag{3.9}$$

Here, F_{com} denotes the set of common features identified based on a similarity threshold τ and a correlation matrix difference threshold ϵ.

- **Feature Aggregation:** The server aggregates the common features from all clients to update the global model:

$$F_{\text{Glob}} = A(F^{(1)}_{\text{common}}, F^{(2)}_{\text{common}}, \ldots, F^{(N)}_{\text{common}}) \tag{3.10}$$

This aggregated feature matrix is used to update the global model, which is then redistributed to the clients for further refinement. The entire process of the Feature Align-HFL framework is in pseudocode detailed in Algorithm 1. At the outset of the training process, both the server and the clients begin without any pre-trained feature representations.

Algorithm 1 Feature Align-HFL Framework

Input : Number of communication rounds T, Number of local epochs E, Number of clients k, Private datasets $D_v = (X_v, Y_v)$ for $v = 1, 2, \ldots, k$

Output: Global model, F_{Glob}

Server Process:

Initialize global model F_{Glob}

for *each round* $i = 1, 2, \ldots, T$ **do**

  Select a subset of clients $\mathcal{V} \subseteq \{1, \ldots, k\}$

  for *each client* $v \in \mathcal{V}$ **do**

    Receive F_{com} from client v

    update F_{Glob} as in Eq. (3.10)

  Transmit updated F_{Glob} to all clients

Client Process:

for *each selected client* $v \in \mathcal{V}$ **do**

  Receive global model F_{Glob} from the server

  for *each epoch* $e = 1, 2, \ldots, E$ **do**

    Train model M_v on private dataset D_v

    for *each data point* $x_j \in X_v$ **do**

      Compute local features $f_v(x_j)$

      Align features $\tilde{f}_v(x_j)$ as in Eq. (3.7)

    Perform $S_c(\tilde{f}_v(x_j), \tilde{f}_v(x_l))$ as in Eq. (3.8)

  Compute common features F_{com} as in Eq. (3.9)

  Transmit F_{com} to the server

Return F_{Glob}

Chapter 4

Experimental Methodology

In this section, we outline the experimental methodology used to evaluate the Feature Align-HFL framework. We describe the data set, the models employed, and the specific training procedures followed for CNN and the U-Net model. In addition, we detail the process for identifying common features between these models.

4.1 Dataset Desciption

For the experimental validation of the Feature Align-HFL framework, we utilized two widely used datasets: CIFAR-10 and MNIST. In the following, we provide an introduction, key features, and significance for each dataset.

4.1.1 CIFAR-10 Dataset

The CIFAR-10 dataset is a collection of images commonly used for image classification tasks. It contains 60,000 color images in 10 different classes, with 50,000 images designated for training and 10,000 for testing.

Key Features

- **Resolution:** Each image is 32×32 pixels with RGB color channels.

- **Class Diversity:** Images are evenly distributed across 10 classes, including animals (e.g., cats, dogs) and vehicles (e.g., cars, trucks).
- **Standardized Preprocessing:** This dataset is widely used to benchmark classification models and provides a standard format for model evaluation.

FIGURE 4.1: Illustration of CIFAR-10 dataset

4.1.2 MNIST Dataset

The MNIST dataset consists of handwritten digits from 0 to 9, with a total of 70,000 grayscale images. The dataset is split into 60,000 images for training and 10,000 for testing, making it a classic benchmark for image classification.

Key Features

- **Resolution:** Each image is 28×28 pixels in grayscale, suitable for simple, low-dimensional image recognition.
- **Class Diversity:** Includes 10 digit classes (0–9), covering a variety of handwriting styles.
- **Standardized Dataset:** Widely used for evaluating algorithms in computer vision, particularly for introductory-level classification tasks.

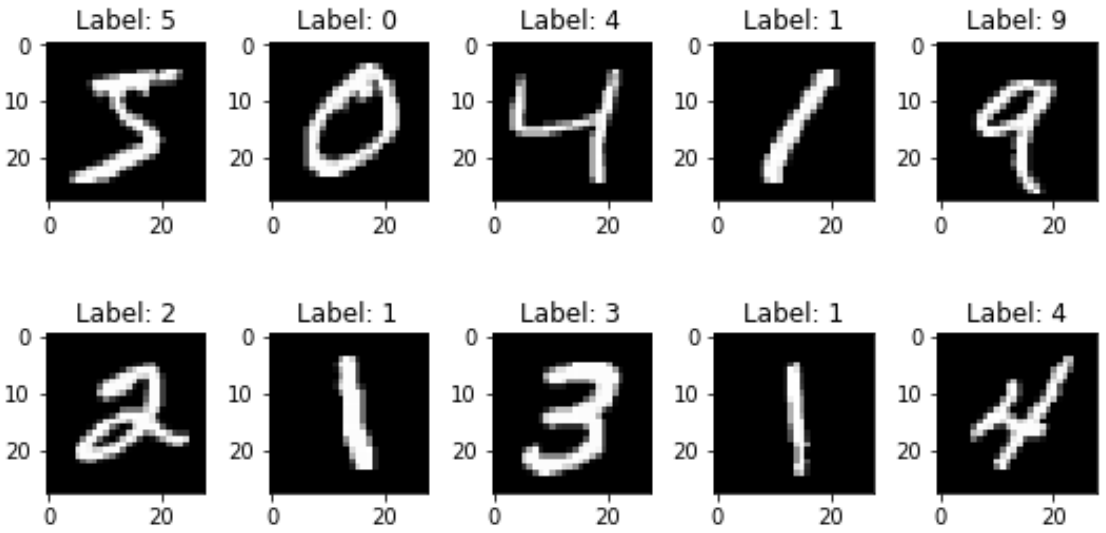

FIGURE 4.2: Illustration of MNIST dataset

4.2 Data Preprocessing

To prepare the CIFAR-10 and MNIST datasets for training and evaluation within the Feature Align-HFL framework, the following preprocessing steps were applied:

- **Dataset Loading:** Both the CIFAR-10 and MNIST datasets were loaded and split into training and testing sets to support model training and performance evaluation.
- **Normalization:** To ensure compatibility with model architectures and improve training stability, the pixel values in both datasets were normalized.
  - For CIFAR-10, each image's pixel values were scaled to a range between 0 and 1 by dividing by 255, which standardized the input data and improved model convergence.
  - The MNIST images, originally in grayscale, were reshaped to include an additional channel dimension, aligning with the network's expected input shape, and then normalized to a range of 0 to 1.

Through these preprocessing steps—loading, normalizing and reshaping the CIFAR-10 and MNIST datasets were prepared in a consistent, standardized format, optimized for effective training and evaluation within the Feature Align-HFL framework.

4.3 Model Architectures

In this section, we provide an overview of the two model architectures used in our experiments: Convolutional Neural Network (CNN) and U-Net. We begin by describing the general concept of each model type, followed by the specific architecture configurations used in our study.

4.3.1 Convolutional Neural Network (CNN)

A Convolutional Neural Network (CNN) is a type of deep learning model specifically designed to process data with a grid-like structure, such as images. CNNs are particularly well-suited for tasks like image classification and object recognition, as they effectively capture spatial and hierarchical features through convolutional operations. The main components of a CNN are convolutional layers, pooling layers, and fully connected layers.

CNN Architecture

A typical CNN architecture consists of an input layer, multiple convolutional layers that apply filters to extract features, pooling layers to reduce spatial dimensions, and finally, fully connected layers for classification. The convolutional layers use filters to identify patterns within the data, such as edges or textures in images, while pooling layers reduce computation by down-sampling the feature maps.

Our CNN Architecture

In our experiments, we used a customized CNN architecture with the following layers:

- **Convolutional Layers:** The architecture begins with two convolutional layers with 32 filters each, followed by additional layers with 64 and 128 filters. Each layer uses ReLU activation functions to introduce non-linearity.
- **Pooling Layers:** Max-pooling layers follow the convolutional layers to progressively down-sample the spatial dimensions.

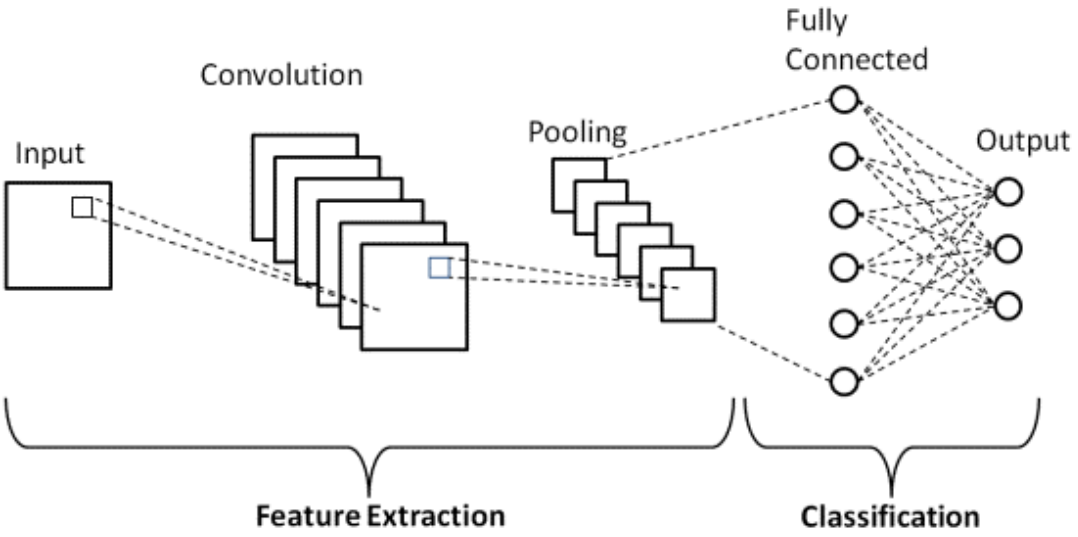

FIGURE 4.3: Basic CNN Architecture

- **Fully Connected Layers:** The network ends with dense layers for classification, transforming the extracted features into output classes.

4.3.2 U-Net

U-Net is a type of convolutional neural network that is particularly effective for image segmentation tasks. Developed with an encoder-decoder structure, U-Net is designed to capture both local and global contextual features, making it suitable for tasks that require high-resolution output, such as biomedical image segmentation.

U-Net Architecture

A typical U-Net architecture consists of two main parts:

- **Encoder:** The encoder is responsible for down-sampling the input image to capture high-level features. It consists of a series of convolutional and pooling layers.
- **Decoder:** The decoder up-samples the encoded features to reconstruct the spatial details and produce an output of the same resolution as the input. Skip connections between corresponding encoder and decoder layers help preserve fine-grained details.

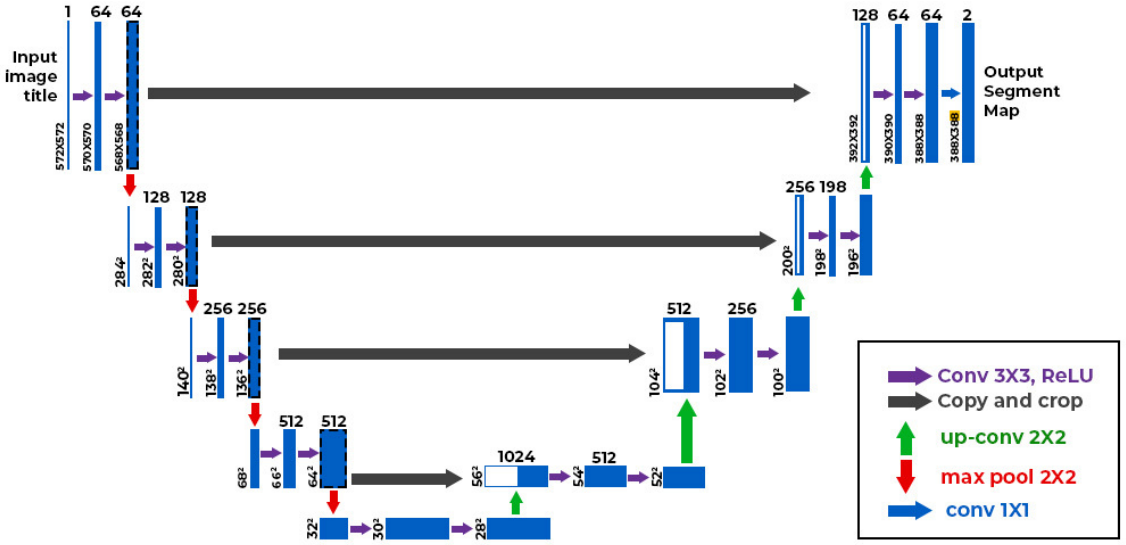

FIGURE 4.4: Basic U-NET Architecture

Our U-Net Architecture

In our implementation, the U-Net model is configured as follows:

- **Encoder:** The encoder portion includes multiple convolutional layers with increasing filter sizes, paired with max-pooling layers for down-sampling.
- **Decoder:** The decoder mirrors the encoder structure, using up-sampling layers to progressively increase spatial resolution.
- **Final Layer:** The final layer outputs classification results, enabling the U-Net to be applied to tasks requiring for classification.

4.4 Experiment

The experiments employed two neural network architectures: a Convolutional Neural Network (CNN) and a U-Net, each selected for its distinct strengths in evaluating the Feature Align-HFL framework. The CNN was chosen for its established effectiveness in image classification tasks, while the U-Net, commonly applied in image segmentation, was adapted here for classification to examine its performance in this area. Both models were trained on subsets of the CIFAR-10 and MNIST datasets,

providing a comprehensive basis for evaluating the Feature Align-HFL framework across diverse image types and classification challenges.

4.4.1 Training Configuration

The models were trained using a learning rate of 0.0001 optimized through the Adam optimizer.The loss function applied during training was categorical cross-entropy, chosen for its suitability in multi-class classification tasks, where it helps in minimizing the difference between the predicted and actual class probabilities.

4.4.2 Performance Evaluation

The performance of the CNN and U-Net models was primarily evaluated in terms of classification accuracy on the CIFAR-10 and MNIST test sets, providing a quantitative measure of their effectiveness in the context of the Feature Align-HFL framework. Beyond accuracy, feature alignment between the models was examined using Principal Component Analysis (PCA) and cosine similarity. These methods enabled a deeper analysis of the learned feature representations, offering insights into how well the models aligned and generalized across varying datasets and architectures.

4.4.3 Machine Specifications

The models were trained and evaluated on a machine with the following specifications:

- **Processor:** Intel Core i7 13gen
- **Graphics Processing Unit (GPU):** Tesla V100-PCIE-32GB
- **Operating System:** Windows 11.
- **Software Libraries:** TensorFlow, Keras.
- **CUDA and cuDNN:** CUDA 11.0 for GPU acceleration.

Chapter 5

Methodology

This chapter presents an overview of the image inpainting project, detailing the design and production choices made during development. The project's working environment, including setup and dataset, is outlined. The chapter covers preprocessing, architecture, training, and testing tasks. A simplified workflow for the model is shown in Figure 5.1. d the position of the lines along with its dimensions.

Chapter 6

Conclusion and Future Scope

6.1 Future Scope

1. Exploring the use of different architectures, such as GANs or transformers, for image inpainting
2. Developing a more efficient and scalable version of the model for large-scale image inpainting tasks
3. Adapting the model for video inpainting applications
4. Conducting a comparative analysis of different deep learning-based inpainting methods on various image datasets

6.2 Conclusion

In this project, we explored various image inpainting methods using deep learning. We used a UNET auto-encoder architecture trained on the CIFAR10 dataset to fill the missing pixels or patches in an image. To simplify the implementation, random lines were inserted within input images at random positions. The corresponding original images were considered labeled images. The implementation was done on Google Colab using GPU for faster processing. Our results demonstrate the effectiveness of this method in efficiently restoring missing information in images, and increasing the number of layers improves model performance.

Chapter 7

Appendices

References