TRƯỜNG ĐẠI HỌC KHOA HỌC TỰ NHIÊN
KHOA CÔNG NGHỆ THÔNG TIN

Thành viên:

21120369 - Nguyễn Minh Vũ

21120558 - Phùng Hoài Thi

Báo cáo thực hành:

HỌC LIÊN KẾT (FEDERATE LEARNING) VÀ BẢO MẬT TẠI SERVER

CSC18101 – Trí tuệ nhân tạo cho an ninh thông tin

Tp. Hồ Chí Minh, tháng 10/2024

Mục lục

Danh sách hình

Danh sách bảng

Nội dung thực hành

Yêu cầu	Phụ trách	Mức độ hoàn thành
Đọc dữ liệu	Vũ	100%
Xử lý dữ liệu	Vũ	100%
Cài đặt mô hình hồi quy logistic	Vũ	100%
Cài đặt mô hình 1 server, 3 clients	Vũ, Thi	100%
Báo cáo về mô hình tấn công Adversarisal Examples	Thi	100%
Viết báo cáo	Vũ, Thi	100%
Làm slide	Vũ, Thi	100%

Bảng 1.1: Bảng đánh giá công việc

Chuẩn bị dữ liệu

Trong bài thực hành về học liên kết (Federate Learning), ta sẽ làm việc trên bộ dữ liệu CIFAR10 - tập dữ liệu hình ảnh được sử dụng rộng rãi cho các thuật toán máy học và thị giác máy tính. CIFAR10 bao gồm 60.000 hình ảnh màu 32×32 pixel chia đều cho 10 lớp, trong đó có 50.000 ảnh cho tập huấn luyện và 10.000 ảnh cho tập kiểm thử. Dưới đây là ví dụ minh hoạ cho 10 lớp:

Hình 2.1: Dữ liệu CIFAR10

Cách tải dữ liệu CIFAR10 về máy:

- Tải thủ công: cifa10-original
- Sử dụng Pytorch và Torchvision: cifar10-pytorch
- Sử dụng Keras: cifar10-keras

Xử lý dữ liệu

Để dữ liệu có thể được sử dụng hiệu quả cho mô hình học Hồi quy Logistic (Logistic Regression), ta sẽ áp dụng phương pháp trích xuất đặc trưng (**Feature Extraction**). Trong đó, 2 hướng tiếp cận chính là sử dụng phương pháp thủ công (**Handcrafted**) và phương pháp áp dụng học sâu (**Deep Learning**).

3.1 Xử lý đặc trưng thủ công HoG

Histogram of Oriented Gradients (HoG) là một phương pháp trích xuất đặc trưng mô tả cấu trúc và hình dạng của đối tượng trong ảnh, được giới thiệu lần đầu vào năm 2005 của Dalal và Triggs trong bài toán phát hiện người đi bộ [1]. Phương pháp này tập trung vào việc tính toán gradient và hướng cạnh tại từng vùng của ảnh.

Đầu tiền nhóm sẽ xử lý ảnh màu thành *ảnh xám (grayscale)* và áp dụng HoG từ thư viện skimage, sau đó cài đặt các tham số sau:

```
normalize = True
block_norm = 'L2-Hys'
orientations = 9
pixels_per_cell = [8, 8]
cells_per_block = [2, 2]
def extractFeature(img, vis=False):
    from skimage.feature import hog
    return hog(img, orientations, pixels_per_cell,
            cells_per_block, block_norm, visualize = vis,
            transform_sqrt=normalize)
```

3.2 Xử lý đặc trưng áp dụng Deep Learning

Trong học sâu, các phương pháp trích xuất đặc trưng tự động, chủ yếu là thông qua các mạng nơ-ron tích chập (CNN - Convolutional Neural Network) [4],[5]. Khác với phương pháp thủ công, Deep Learning cho phép trích xuất đặc trưng từ dữ liệu thô mà không cần xác định đặc trưng cụ thể trước. Đây là kết quả của quá trình học của mô hình từ chính dữ liệu trong quá trình huấn luyện, giúp mô hình học cách nhận biết các mẫu phức tạp từ dữ liệu lớn.

Một trong những cải tiến của kiến trúc mạng CNN là ResNet(Residual Network) [2]. Kiến trúc này giúp giải quyết vấn đề vanishing gradient và degradation - những hạn chế khi làm việc với Deep Neuron Network.

Nhóm sử dụng Keras để thiết lập model Resnet. Resnet50 sẽ được chọn cho bài lab này vì Resnet50 tương đối nhẹ (98MB, 25.6M tham số) và thời gian thực thi tương đối nhanh mà vẫn đảm bảo độ chính xác trên 70%. Quá trình trích xuất đặc trưng sử dụng Resnet như bên dưới:

```
def preprocess_image_input(input_images):
  input_images = input_images.astype('float32')
  input_images = tf.keras.layers.UpSampling2D(size=(sampling,
  sampling))(input_images)
  output_ims = preprocess_input(input_images)
  return output_ims
fe_model = ResNet50(weights='imagenet',
    include_top=False, input_shape=(image_size[0] * sampling,
    image_size[1] * sampling, image_size[2]))
```

Đầu tiên, dữ liệu sẽ được tiền xử lý về đúng dạng dữ liệu cho mô hình. Sau đó, UpSampling2D sẽ được áp dụng để làm giàu dữ liệu (ban đầu ảnh chỉ có kích thước $32 \times 32 \times 3$, khi dùng UpSampling2D với kích thước bằng 2 thì ta sẽ tăng gấp đôi số pixel của ảnh). Hàm *preprocess_input* của Resnet cũng được áp dụng để chuẩn hoá dữ liệu. Sau khi đã tiền xử lý, ta sẽ đưa dữ liệu qua mô hình để trích xuất đặc trưng.

Mô hình

4.1 Logistic Regression

Logistic Regression hay mô hình hồi quy Logistic là mô hình được sử dụng phổ biến cho các bài toán phân lớp nhị phân (2-lớp) bên cạnh Linear Regression. Bên cạnh đó, ta cũng có các biến thể có thể kể đến như Softmax Regression giúp phân loại đa lớp (nhiều hơn 3 lớp).

Nhóm sử dụng thư viện Keras và Tensorflow để xây dựng mô hình học máy Logistic Regression. Cấu trúc mô hình như sau:

```
def create_model(n_features):
    """Creates a Logistic Regression model."""
    model = Sequential([
        Input(shape=(n_features)),
        Flatten(),
        Dense(10, activation='softmax')
    ])
    model.compile(optimizer=Adam(learning_rate=LR),
    loss=SparseCategoricalCrossentropy(from_logits=False),
    metrics=['accuracy'])
    return model
```

Trong đó, cấu hình tham số nhóm sử dụng gồm phương pháp tối ưu Adam, hàm mất mát Sparse Categorical Crossentropy và đo lường độ chính xác (accuracy) trong quá trình huấn luyện và đánh giá.

4.2 Huấn luyện mô hình

Nhóm sử dụng các phương pháp trích xuất đặc trưng khác nhau để kiểm thử độ chính xác của mô hình và sử dụng checkpoint để lưu model có trọng số tốt nhất. Dưới đây là cách nhóm huấn luyện và kiểm thử mô hình:

```
    checkpoint = ModelCheckpoint(
    'model/model.keras',# Path where to save the model
    monitor='val_accuracy',# Metric to monitor
    save_best_only=True,# Save only the best model
    mode='max',
    verbose=1
)
model.fit(X_training, y_training, epochs=num_epochs,
batch_size=batch_size, validation_data=(X_testing, y_testing
, callbacks=[checkpoint])
```

4.3 Kết quả dự đoán của mô hình

4.3.1 Không sử dụng phương pháp trích xuất đặc trưng

Class	**Precision**	**Recall**	**F1-score**	**Support**
Airplane	0.2406	0.1470	0.1825	1000
Automobile	0.6081	0.0900	0.1568	1000
Bird	0.4375	0.0140	0.0271	1000
Cat	0.2209	0.3130	0.2590	1000
Deer	1.0000	0.0030	0.0060	1000
Dog	0.3125	0.0250	0.0463	1000
Frog	0.3538	0.4490	0.3958	1000
Horse	0.4610	0.3190	0.3771	1000
Ship	0.2062	0.8590	0.3326	1000
Truck	0.3190	0.5050	0.3910	1000
Accuracy	0.2724			10000
Macro Avg	0.4160	0.2724	0.2174	10000
Weighted Avg	0.4160	0.2724	0.2174	10000

Bảng 4.1: Bảng đánh giá không dùng đặc trưng

Như ta có thể thấy, khi không sử dụng phương pháp trích xuất đặc trưng cho bài toán phân loại đa lớp (10-lớp) thì mô hình Logistic Regression sẽ không hiệu quả vì tính chất của Logistic với softmax là phân lớp dựa trên xác suất. Vì dữ liệu ban đầu không được xử lý mà được đưa thẳng qua mô hình nên xác suất của mỗi lớp bị nhiễu dẫn đến độ chính xác không cao.

4.3.2 Sử dụng đặc trưng HoG

Để cải thiện mức độ chính xác của mô hình, ta có thể áp dụng phương pháp HoG để trích xuất đặc trưng từ ảnh gốc. Ta thấy độ chính xác của mô hình đã tăng lên $\approx 50\%$.

Class	**Precision**	**Recall**	**F1-score**	**Support**
Airplane	0.5961	0.5800	0.5879	1000
Automobile	0.5981	0.6190	0.6084	1000
Bird	0.4433	0.3870	0.4132	1000
Cat	0.4365	0.2820	0.3426	1000
Deer	0.4752	0.3920	0.4296	1000
Dog	0.4210	0.4530	0.4364	1000
Frog	0.4677	0.6650	0.5491	1000
Horse	0.5255	0.6080	0.5637	1000
Ship	0.5921	0.5850	0.5885	1000
Truck	0.6289	0.6320	0.6304	1000
Accuracy	0.5203			10000
Macro Avg	0.5184	0.5203	0.5150	10000
Weighted Avg	0.5184	0.5203	0.5150	10000

Bảng 4.2: Bảng đánh giá với đặc trưng HoG

4.3.3 Sử dụng đặc trưng Resnet

Ở phương pháp HoG, ta đã đạt được độ chính xác khoảng 50%. Tuy đã cải thiện hơn ban đầu, nhưng đến đây vẫn chưa đủ. Việc mô hình dự đoán sai 50% vẫn là tương đối ảnh hưởng (100 ảnh thì tỉ lệ là dự đoán sai 50 ảnh và đúng 50 ảnh). Điều này dẫn ta đến một cách tiếp cận nâng cao hơn, đó là sử dụng mạng tích chập CNN từ Deep Learning, cụ thể hơn là mô hình Resnet:

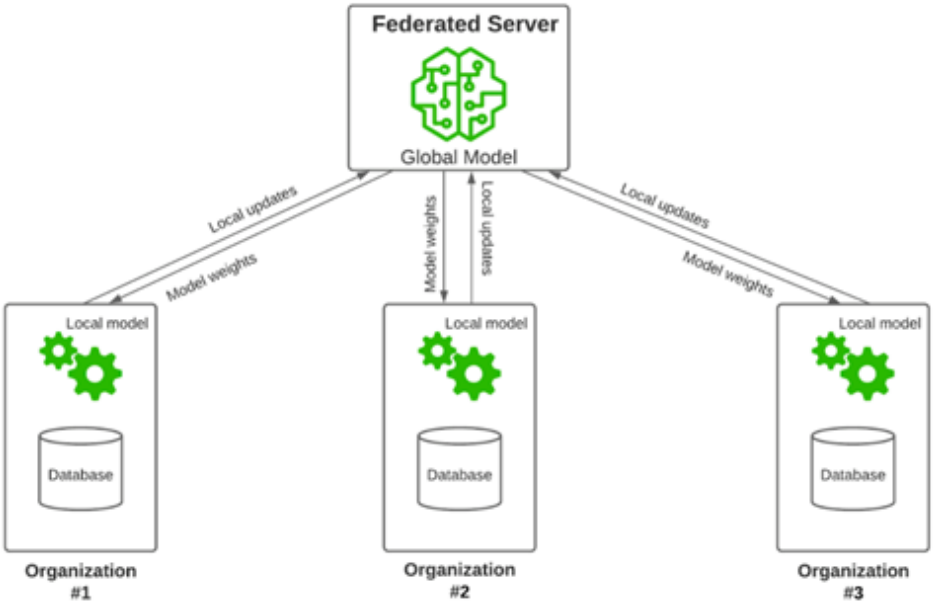
Class	Precision	Recall	F1-score	Support
Airplane	0.7610	0.8470	0.8017	1000
Automobile	0.8526	0.8910	0.8714	1000
Bird	0.7417	0.7380	0.7398	1000
Cat	0.6647	0.5750	0.6166	1000
Deer	0.7546	0.6580	0.7030	1000
Dog	0.7453	0.7110	0.7277	1000
Frog	0.7581	0.8870	0.8175	1000
Horse	0.7323	0.8700	0.7952	1000
Ship	0.8734	0.8420	0.8574	1000
Truck	0.9269	0.7730	0.8430	1000
Accuracy	0.7792			10000
Macro Avg	0.7811	0.7792	0.7773	10000
Weighted Avg	0.7811	0.7792	0.7773	10000

Bảng 4.3: Bảng đánh giá với đặc trưng Resnet

Kết quả dự đoán của mô hình với đặc trưng Resnet $\approx 78\%$, tuyệt vời. Tuy nhiên, ta vẫn có thể tăng độ chính xác của mô hình thông qua việc làm giàu dữ liệu bằng UpSampling2D. Nhóm quyết định dừng lại ở việc Sampling gấp đôi ảnh gốc vì ta vẫn còn nhiều công đoạn ở phía trước.

Federated Learning

Federated Learning (FL), hay học liên kết, là một phương pháp học máy phân tán trong đó dữ liệu không được tập trung tại một máy chủ trung tâm mà thay vào đó được giữ tại các thiết bị đầu cuối (client). Trong FL, các mô hình được đào tạo trên các thiết bị riêng lẻ và chỉ các cập nhật mô hình (như trọng số hoặc gradient) được gửi về máy chủ trung tâm để tổng hợp, giữ dữ liệu nhạy cảm trên client và tăng tính bảo mật cho người dùng. [6]



Hình 5.1: Mô tả Federate Learning

5.1 Tại sao cần Federated Learning?

Federated Learning ra đời nhằm giải quyết các vấn đề chính sau:

- **Bảo vệ quyền riêng tư dữ liệu**: Với FL, dữ liệu người dùng không cần phải rời khỏi thiết bị cá nhân, giúp giảm thiểu rủi ro bị đánh cắp

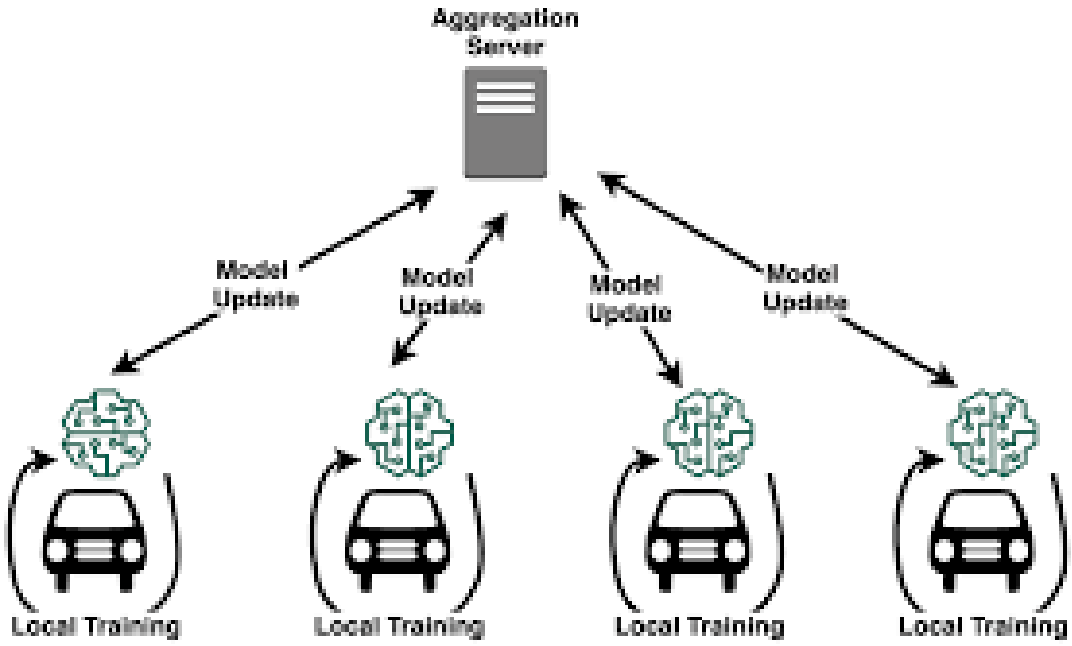
hoặc rò rỉ dữ liệu.

- **Giảm tải băng thông**: Việc chỉ gửi các trọng số thay vì dữ liệu thô giúp giảm lưu lượng mạng.

5.2 Ứng dụng của Federated Learning

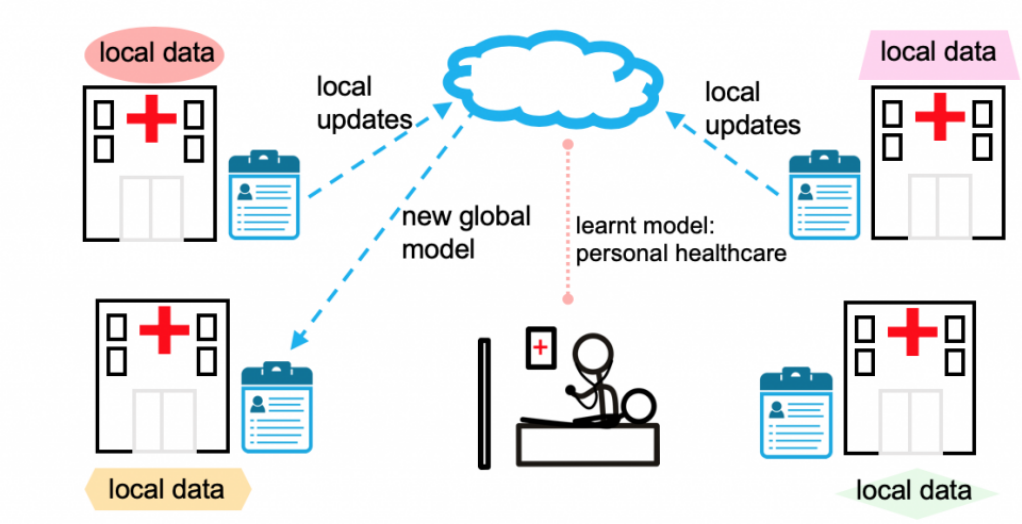
FL được ứng dụng trong nhiều lĩnh vực khác nhau, đặc biệt là trong các tình huống yêu cầu bảo mật dữ liệu cao, bao gồm:

- **Ô tô tự lái**: FL giúp các công ty phát triển công nghệ ô tô tự lái huấn luyện các mô hình trên dữ liệu thu thập từ xe mà không cần tải về trung tâm, tiết kiệm băng thông và bảo mật dữ liệu người dùng.



Hình 5.2: Federate Learning trong xe tự lái

- **Y tế và chăm sóc sức khỏe**: FL cho phép các bệnh viện hoặc cơ sở y tế hợp tác trong việc huấn luyện mô hình mà không cần chia sẻ dữ liệu bệnh nhân, giúp tuân thủ các quy định về bảo mật như HIPAA.



Hình 5.3: Federate Learning trong y tế

5.3 Rủi ro của Federated Learning

Mặc dù FL có nhiều lợi ích, nó cũng tồn tại một số rủi ro và thách thức:

- **Rủi ro tấn công và rò rỉ thông tin**: Mặc dù dữ liệu không rời khỏi thiết bị, kẻ tấn công vẫn có thể lợi dụng các cập nhật mô hình để trích xuất thông tin nhạy cảm của người dùng.
- **Chi phí tính toán cao trên thiết bị**: Quá trình huấn luyện mô hình trên thiết bị cuối yêu cầu sức mạnh tính toán, đặc biệt đối với các thiết bị di động có cấu hình thấp.
- **Thách thức trong việc tổng hợp mô hình**: Quá trình tổng hợp các mô hình cục bộ từ các thiết bị có thể gặp khó do sự không đồng nhất của dữ liệu.

5.4 Học Đồng Bộ và Học Bất Đồng Bộ

5.4.1 Học Đồng Bộ

Trong học đồng bộ, server sẽ đợi tất cả các client hoàn thành một số batch huấn luyện nhất định trước khi thực hiện tổng hợp trọng số. Điều

này giúp đảm bảo tính nhất quán và đồng nhất của mô hình, tuy nhiên có thể gây độ trễ nếu một hoặc một vài client chậm hơn các client còn lại.

Quy trình giao tiếp trong Học Đồng Bộ

Dưới đây là các bước cụ thể trong quá trình học đồng bộ giữa server và client:

Algorithm 1 Quy trình Học Đồng Bộ trong Federated Learning

1: **Server:** Khởi tạo mô hình toàn cục $\mathbf{W}_0$ và gửi đến tất cả các client.
2: **for** each round $t = 1, 2, \ldots, T$ **do**
3: **Server:** Gửi bộ trọng số mới nhất cho các client.
4: **Client:** Nhận yêu cầu và tải mô hình $\mathbf{W}_t$ từ server.
5: **Client:** Huấn luyện mô hình $\mathbf{W}_t$ trên dữ liệu cục bộ trong N batch, kết quả là mô hình cục bộ $\mathbf{W}_t^i$.
6: **Client:** Gửi $\mathbf{W}_t^i$ về server.
7: **Server:** Đợi cho đến khi nhận đủ trọng số từ tất cả các client.
8: **Server:** Tổng hợp các trọng số để cập nhật mô hình toàn cục:

$$\mathbf{W}_{t+1} = \frac{1}{N} \sum_{i=1}^{N} \mathbf{W}_t^i \tag{5.1}$$

9: **Server:** Lặp lại quá trình từ bước 2.

Ưu và nhược điểm

- **Ưu điểm:** Đảm bảo tính nhất quán giữa các lần tổng hợp, mô hình có sự đồng nhất ở mỗi vòng huấn luyện.
- **Nhược điểm:** Độ trễ cao khi có client hoạt động chậm, làm giảm tốc độ toàn hệ thống.

5.4.2 Học Bất Đồng Bộ

Trong học bất đồng bộ, server không cần đợi tất cả các client hoàn thành huấn luyện. Khi bất kỳ client nào hoàn thành quá trình huấn luyện cục bộ, trọng số sẽ được gửi về server để tổng hợp ngay lập tức. Phương

pháp này giúp giảm thiểu độ trễ nhưng có thể dẫn đến thiếu ổn định do sự không đồng bộ trong cập nhật trọng số.

Quy trình giao tiếp trong Học Bất Đồng Bộ

Các bước trong quá trình học bất đồng bộ được mô tả như sau:

Algorithm 2 Quy trình Học Bất Đồng Bộ trong Federated Learning

1: **Server:** Khởi tạo mô hình toàn cục $\mathbf{W}_0$ và gửi đến các client.
2: **Server:** Gửi yêu cầu huấn luyện đến tất cả client.
3: **loop**
4: **Client:** Nhận yêu cầu và tải mô hình $\mathbf{W}_t$ từ server.
5: **Client:** Huấn luyện mô hình $\mathbf{W}_t$ trên dữ liệu cục bộ trong N batch, tạo ra mô hình cục bộ $\mathbf{W}_t^i$.
6: **Client:** Gửi $\mathbf{W}_t^i$ về server ngay khi hoàn thành huấn luyện.
7: **Server:** Nhận trọng số từ các client và cập nhật mô hình toàn cục ngay lập tức:

$$\mathbf{W}_{t+1} = \mathbf{W}_t + \eta \cdot (\mathbf{W}_t^i - \mathbf{W}_t) \tag{5.2}$$

8: **Server:** Tiếp tục lặp khi có client mới gửi trọng số về.

Ưu và nhược điểm

- Ưu điểm: Giảm thiểu độ trễ khi các client hoàn thành huấn luyện tại các thời điểm khác nhau.
- Nhược điểm: Có thể dẫn đến sự không ổn định của mô hình nếu các client có tốc độ huấn luyện quá khác biệt.

5.5 Các Phương Pháp Tổng Hợp Chính

Trong lĩnh vực Học Liên Kết, các phương pháp tổng hợp (aggregation methods) đóng vai trò quan trọng trong việc kết hợp thông tin từ nhiều client khác nhau để xây dựng mô hình toàn cục. [6]

5.5.1 FedAvg

FedAvg là một trong những phương pháp đầu tiên và phổ biến nhất trong FL. Phương pháp này thực hiện tổng hợp bằng cách tính trung bình

có trọng số các mô hình từ các client. Trọng số được xác định dựa trên số lượng dữ liệu mà mỗi client sở hữu. Công thức tổng hợp trong FedAvg được biểu diễn như sau:

$$w_{\text{glob}}^{t+1} = \sum_{k \in S_t} \frac{n_k}{n} w_k^{t+1}$$

Trong đó:

- w_{glob}^{t+1} là mô hình toàn cục sau khi cập nhật.
- S_t là tập hợp các client được chọn tại vòng lặp t.
- n_k là số lượng dữ liệu của client k.
- n là tổng số dữ liệu của tất cả các client.

Phương pháp này giúp đảm bảo rằng các client có nhiều dữ liệu hơn sẽ có ảnh hưởng lớn hơn đến mô hình toàn cục.

5.5.2 FedProx

FedProx là một biến thể của FedAvg, nhằm giải quyết vấn đề tối ưu hóa cục bộ và sự không đồng nhất giữa các client. Để điều chỉnh sự ảnh hưởng của các mô hình cục bộ, FedProx bổ sung một thuật ngữ điều chỉnh vào hàm mục tiêu của mỗi client. Công thức cho hàm mục tiêu trong FedProx được định nghĩa như sau:

$$\mathcal{L}_{\text{prox}} = \mathcal{L}(w_k) + \frac{\mu}{2} \|w_k^t - w_{\text{glob}}^t\|^2$$

Trong đó:

- $\mathcal{L}(w_k)$ là hàm mất mát của client k.
- μ là một hằng số điều chỉnh, cho phép kiểm soát độ lớn của thuật ngữ điều chỉnh.
- w_k^t là mô hình cục bộ của client k tại vòng lặp t.
- w_{glob}^t là mô hình toàn cục tại vòng lặp t.

Hàm điều chỉnh này giúp các client điều chỉnh các cập nhật của họ gần hơn với mô hình toàn cục, từ đó giảm thiểu sai số trong quá trình học.

5.5.3 FedNova

FedNova là một cải tiến của FedAvg, trong đó các cập nhật từ mỗi client được chuẩn hóa và điều chỉnh dựa trên số vòng lặp địa phương mà mỗi client thực hiện. Điều này giúp tối ưu hóa việc học từ các client khác nhau. Công thức cập nhật trong FedNova được biểu diễn như sau:

$$w^{t+1} = w_{\text{glob}} - \tau_{\text{eff}} \sum_{k \in S_t} \frac{n_k}{n} \cdot \eta d_k^t$$

Trong đó:

- τ_{eff} là bước lặp hiệu quả, thể hiện sự ảnh hưởng của các cập nhật.
- η là tỷ lệ học (learning rate).
- d_k^t là gradient sau khi được chuẩn hóa của client k tại vòng lặp t.

Phương pháp này cho phép mô hình toàn cục học hỏi tốt hơn từ các cập nhật của từng client, ngay cả khi các client có số vòng lặp khác nhau.

5.5.4 Scaffold

Scaffold là một phương pháp tổng hợp sử dụng biến kiểm soát để giảm thiểu sai lệch do dữ liệu không đồng nhất giữa các client. Biến kiểm soát này được cập nhật dựa trên sự khác biệt giữa mô hình toàn cục và mô hình cục bộ của các client. Công thức cho biến kiểm soát được định nghĩa như sau:

$$c_k^+ = c_k - c + \frac{1}{\eta l}(w_{\text{glob}}^t - w_k^t)$$

Trong đó:

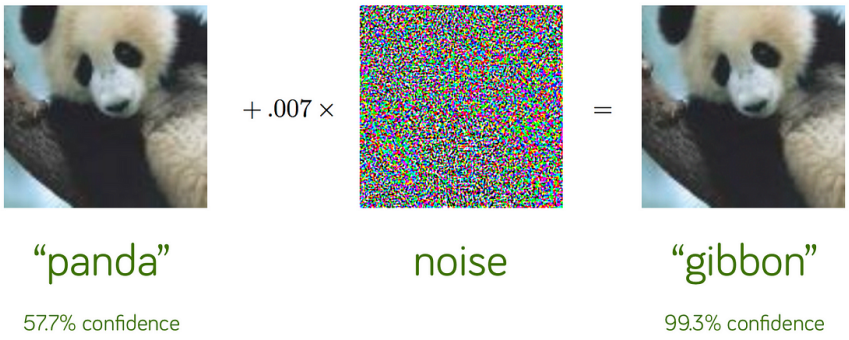
- c_k là biến kiểm soát của client k.
- c là biến kiểm soát trung bình.

- η là tỷ lệ học.
- l là số lần lặp (iterations) của client.

Việc sử dụng biến kiểm soát giúp các client đồng bộ hóa và giảm thiểu ảnh hưởng của sự không đồng nhất trong dữ liệu, từ đó cải thiện khả năng tổng hợp mô hình toàn cục. [3]

5.6 Gradient Ascent cho Tấn công Đối kháng

Trong kịch bản tấn công đối kháng dạng hộp đen (black-box), kẻ tấn công cố gắng tạo ra một đầu vào có thể khiến mô hình mục tiêu phân loại sai. Đầu vào này nhìn qua có vẻ gần như không khác so với ảnh gốc nhưng có thể làm tăng lỗi dự đoán của model.



Hình 5.4: Ví dụ làm nhiễu ảnh

Mục tiêu

Mục tiêu của tấn công là tạo ra một nhiễu δ sao cho đầu vào bị nhiễu $x^{adv} = x + \delta$ khiến mô hình phân loại sai. Điều này được thực hiện bằng cách tối đa hóa hàm mất mát $J(\theta, x, y)$ thông qua thuật toán Gradient Ascent để tăng khả năng dự đoán sai của mô hình. Ký hiệu:

- x: đầu vào ban đầu.
- y: nhãn mục tiêu mong muốn.
- δ: nhiễu đối kháng.

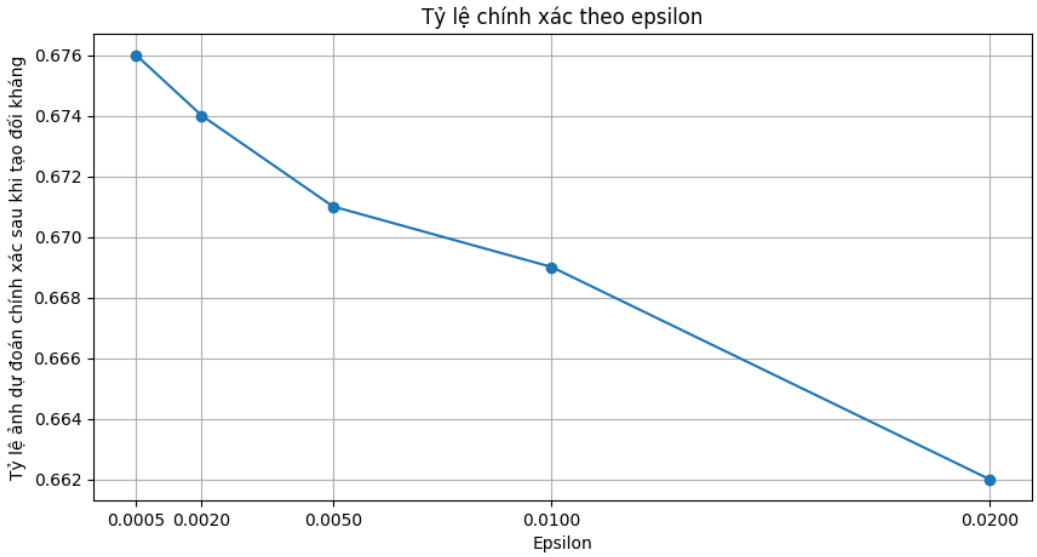
- α: hệ số học (learning rate) cho bước gradient ascent.

Algorithm 3 Tạo mẫu đối kháng bằng thuật toán Gradient Ascent

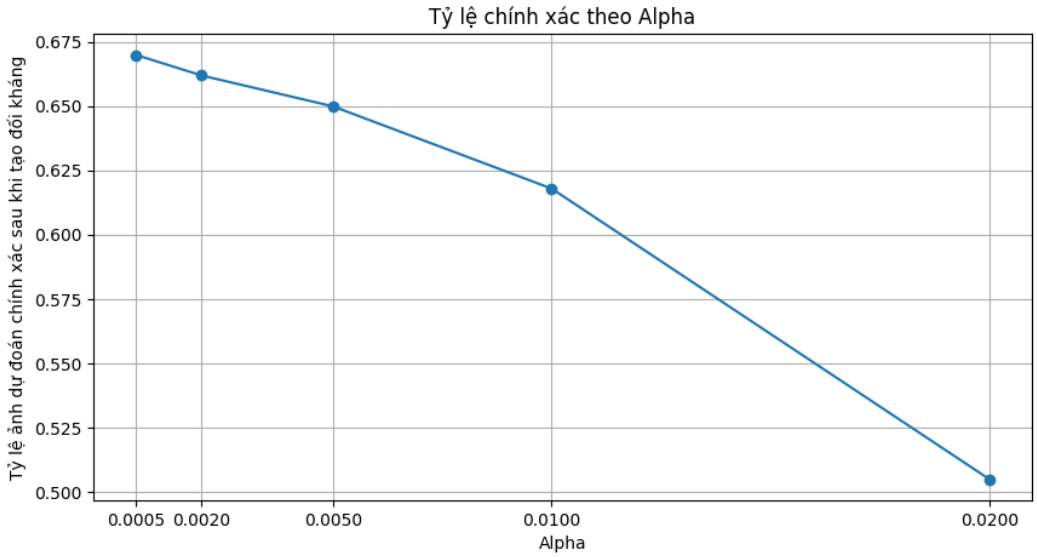
Khởi tạo: Đầu vào x, nhãn y, hệ số học α, số lần lặp N.
Khởi tạo nhiễu $\delta \leftarrow 0$.
for $i = 1$ **to** N **do**
    Tính gradient: $g \leftarrow \nabla_x J(\theta, x + \delta, y)$ ▷ Tính gradient hàm mất mát
    Cập nhật nhiễu: $\delta \leftarrow \delta + \alpha \cdot SIGN(g)$ ▷ Cập nhật nhiễu theo hướng gradient ascent
    Chuẩn hóa δ để đảm bảo $x + \delta$ hợp lệ.
$x^{adv} \leftarrow x + \delta$ ▷ Mẫu đối kháng
return x^{adv}

Mô hình Tấn công

Để tấn công vào mô hình mục tiêu ResNet + Softmax, giả sử ta khám phá được mô hình của mình dùng deep learning để xử lý đặc trưng và softmax để phân loại. Ta xây dựng một mô hình tương đương là VGG16 + Softmax để mô phỏng hành vi, từ đó ước tính gradient $\nabla_x J(\theta, x, y)$ và thực hiện cập nhật nhiễu.



Hình 5.5: LGSM-Epsilon



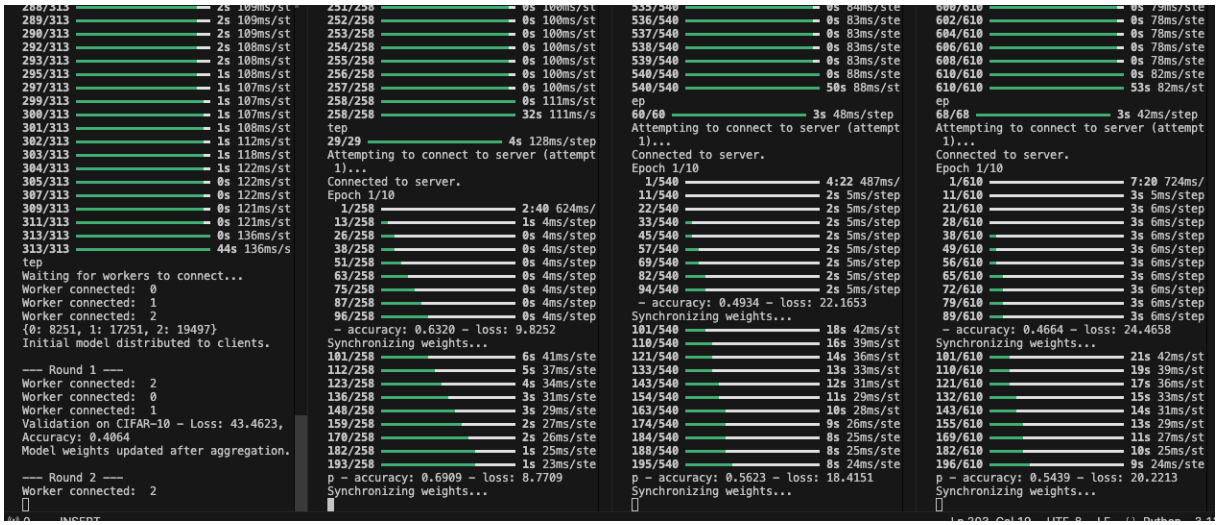
Hình 5.6: LGSM-Alpha

Thử nghiệm

6.1 Thực thi chương trình

Tạo 4 command prompt độc lập để chạy mô hình server-client:

- Server: Chạy lệnh "*python server.py*", server load dữ liệu test để tính độ lỗi trọng số từ các mô hình client.
- Client: Chạy lệnh "*python client.py –client_index* $(0, 1, 2)$*"* tương đương với 3 prompt còn lại để load, huấn luyện mô hình và cập nhật trọng số cho server tổng hợp.



Hình 6.1: Quá trình học liên kết

6.2 Đánh giá với mô hình độc lập

Mô hình	Độ chính xác
Federate Learning	0.7896
Resnet + Logistic	0.7947
HoG + Logistic	0.5173
Logistic	0.2724

Bảng 6.1: Bảng so sánh các mô hình học

Tài liệu tham khảo

[1] Navneet Dalal and Bill Triggs. "Histograms of oriented gradients for human detection". In: *2005 IEEE computer society conference on computer vision and pattern recognition (CVPR'05)*. Vol. 1. Ieee. 2005, pp. 886–893.

[2] Kaiming He et al. "Deep residual learning for image recognition". In: *Proceedings of the IEEE conference on computer vision and pattern recognition*. 2016, pp. 770–778.

[3] Sai Praneeth Karimireddy et al. *SCAFFOLD: Stochastic Controlled Averaging for Federated Learning*. 2021. arXiv: 1910.06378 [cs.LG]. URL: https://arxiv.org/abs/1910.06378.

[4] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems* 25 (2012).

[5] Yann LeCun et al. "Backpropagation applied to handwritten zip code recognition". In: *Neural computation* 1.4 (1989), pp. 541–551.

[6] Pian Qi et al. "Model aggregation techniques in federated learning: A comprehensive survey". In: *Future Generation Computer Systems* 150 (2024), pp. 272–293. ISSN: 0167-739X. DOI: https://doi.org/10.1016/j.future.2023.09.008. URL: https://www.sciencedirect.com/science/article/pii/S0167739X23003333.