ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH

TRƯỜNG ĐẠI HỌC CÔNG NGHỆ THÔNG TIN

KHOA HỆ THỐNG THÔNG TIN

BÁO CÁO BÀI TẬP ĐỒ ÁN

ĐỀ TÀI: SECURITY MACHINE LEARNING LABEL FLIPPING ATTACK

LỚP: IS335.P11.HTCL

Giáo viên hướng dẫn:

Ths. Hà Lê Hoài Trung

Sinh viên thực hiện:

Nguyễn Thế Vinh - 21522794

Chế Duy Khang – 21522187

Trương Vĩnh Thuận – 21522653

Bùi Văn Thái – 21522577

THÀNH PHỐ HỒ CHÍ MINH, 2024

MỤC LỤC

DANH MỤC HÌNH

1. GIỚI THIỆU VẤN ĐỀ

1.1. Giới thiệu bài toán

Trong lĩnh vực học máy và an ninh thông tin, **data poisoning** là một kỹ thuật tấn công nguy hiểm, trong đó **kẻ tấn công cố ý can thiệp vào dữ liệu huấn luyện** của các mô hình học máy nhằm gây ra sai lệch, làm suy yếu hoặc thậm chí kiểm soát hoạt động của mô hình. Mục tiêu của kỹ thuật này là làm giảm hiệu suất, độ chính xác, và độ tin cậy của mô hình, khiến nó không còn hoạt động như mong đợi.

Data poisoning thường được thực hiện bằng cách chèn các mẫu dữ liệu độc hại, bị bóp méo hoặc giả mạo vào tập dữ liệu huấn luyện. Những dữ liệu này có thể được thiết kế cẩn thận để dẫn đến những hậu quả nghiêm trọng, như việc mô hình tạo ra các dự đoán sai lầm hoặc đưa ra các kết quả không phù hợp trong các tình huống quan trọng. Chẳng hạn, trong lĩnh vực nhận dạng hình ảnh, các mẫu bị nhiễm độc có thể khiến mô hình phân loại sai một đối tượng nguy hiểm như vũ khí thành vật vô hại, hoặc trong các hệ thống phát hiện gian lận, kẻ tấn công có thể tạo ra các lỗ hổng để vượt qua kiểm tra bảo mật.

Ngoài ra, data poisoning không chỉ gây ra tổn thất về mặt hiệu suất mà còn có thể dẫn đến hậu quả về bảo mật và quyền riêng tư. Ví dụ, mô hình bị nhiễm độc có thể bị khai thác để làm rò rỉ thông tin nhạy cảm hoặc tạo ra các lỗ hổng bảo mật mà kẻ tấn công có thể lợi dụng. Do đó, data poisoning là một thách thức lớn trong việc bảo đảm tính toàn vẹn của dữ liệu và độ tin cậy của các hệ thống học máy, đặc biệt là trong các lĩnh vực nhạy cảm như tài chính, y tế, và quân sự.

Các biện pháp phòng chống kỹ thuật này bao gồm việc **làm sạch dữ liệu, phát hiện và loại bỏ các mẫu độc hại**, cũng như sử dụng các thuật toán học máy có khả năng chống chịu trước sự can thiệp ác ý. Tuy nhiên, đây vẫn là một bài toán khó vì kẻ tấn công ngày càng phát triển các phương pháp tinh vi hơn, đòi hỏi các nhà nghiên cứu và kỹ sư an ninh không ngừng nâng cao năng lực và cập nhật các chiến lược phòng thủ hiệu quả.

1.2. Dataset

CIFAR-10 (Canadian Institute for Advanced Research) là một trong những bộ dữ liệu phổ biến nhất trong lĩnh vực học máy.

Bộ dữ liệu chứa 60.000 hình ảnh màu, chia thành 10 lớp khác nhau. Mỗi lớp chứa 6.000 hình ảnh, với tổng số 10 lớp tương ứng với các đối tượng khác nhau.

Kích thước hình ảnh: Mỗi hình ảnh có kích thước 32x32 pixel và 3 kênh màu (RGB).

Số lớp: CIFAR-10 có 10 lớp, mỗi lớp đại diện cho một loại đối tượng, bao gồm: Plane, Car, Bird, Cat, Deer, Dog, Frog, Horse, Ship và Truck.

Dữ liệu huấn luyện và kiểm thử:

- 50.000 hình ảnh huấn luyện: 5.000 hình ảnh cho mỗi lớp.
- 10.000 hình ảnh kiểm thử: 1.000 hình ảnh cho mỗi lớp.
- Tất cả các hình ảnh đều được đánh dấu với nhãn tương ứng

2. CƠ SỞ LÝ THUYẾT

2.1. Lý thuyết loại tấn công lựa chọn:

2.1.1. Khái niệm

Label Flipping Attack là một hình thức tấn công dữ liệu đào tạo (training data poisoning) trong đó kẻ tấn công cố ý thay đổi nhãn của một số mẫu dữ liệu để làm giảm hiệu suất của mô hình học máy.

- Dữ liệu bị thay đổi (poisoned data): Một phần nhỏ dữ liệu đào tạo bị thay đổi nhãn.
- Mục tiêu của tấn công: Khi mô hình được đào tạo trên dữ liệu bị nhiễm, nó có thể đưa ra các dự đoán sai hoặc hoạt động kém trên dữ liệu kiểm tra.

2.1.2. Mô hình hoạt động

- Chọn mẫu cần tấn công:

    - Kẻ tấn công xác định một số mẫu trong tập dữ liệu đào tạo để thay đổi nhãn.
    - Các mẫu được chọn thường nằm gần biên phân loại (decision boundary) để tăng hiệu quả tấn công.
- Đảo nhãn (Flip labels):
    - Đổi nhãn của mẫu từ nhãn đúng sang một nhãn sai khác:
        - Ví dụ: Trong bài toán nhị phân: Đổi nhãn từ 0 thành 1 hoặc ngược lại. Trong bài toán đa lớp: Đổi nhãn từ lớp A sang một lớp khác (cố ý hoặc ngẫu nhiên).
- Đào tạo mô hình:
    - Dữ liệu đào tạo bị nhiễm độc được sử dụng để huấn luyện mô hình.
    - Kết quả là mô hình học được thông tin sai lệch, dẫn đến hiệu suất kém.

2.1.3. Tác động

- Hiệu suất giảm sút
    - Mô hình bị đào tạo trên dữ liệu bị nhiễm sẽ không hoạt động tốt trên tập kiểm tra hoặc trong các ứng dụng thực tế.
    - Trong các bài toán phân loại, tỉ lệ lỗi phân loại (misclassification rate) tăng lên rõ rệt.
- Ngụy tạo biên phân loại sai
    - Dữ liệu bị nhiễm độc khiến mô hình học biên phân loại (decision boundary) sai lệch, làm giảm khả năng phân biệt các lớp.
- Tấn công nhắm mục tiêu (Targeted attack):
    - Kẻ tấn công có thể chọn một mục tiêu cụ thể (ví dụ: một mẫu, một lớp) để làm giảm hiệu suất trên mục tiêu này mà không làm giảm hiệu suất tổng thể, tránh bị phát hiện.

2.1.4. Phân loại Label Flipping Attack

- Random Label Flipping:
    - Đảo nhãn ngẫu nhiên của một tỷ lệ nhỏ các mẫu trong dữ liệu đào tạo.
    - Thường ít hiệu quả hơn so với tấn công có mục tiêu, nhưng dễ thực hiện.
- Targeted Label Flipping:
    - Chọn lọc các mẫu cụ thể, thường là các mẫu gần biên phân loại hoặc các mẫu quan trọng trong tập dữ liệu.
    - Mục tiêu là tối đa hóa ảnh hưởng của tấn công trong khi giữ nguyên hiệu suất tổng thể, để tránh bị phát hiện.

2.1.5. Ví dụ minh họa

Ví dụ trong bài toán nhị phân

- Dữ liệu gốc: Mẫu x1 có nhãn đúng là 0. Mẫu x2 có nhãn đúng là 1.
- Tấn công: Đổi nhãn x1 thành 1 và x2 thành 0.
- Kết quả: Mô hình học sai rằng x1 thuộc lớp 1 và x2 thuộc lớp 0, dẫn đến biên phân loại sai lệch.

2.2. Model sử dụng trong cài đặt chương trình:

ResNet32 là một mạng nơ-ron sâu được sử dụng phổ biến trong phân loại hình ảnh, đặc biệt trên các bộ dữ liệu như CIFAR-10. Nó là một phiên bản của kiến trúc Residual Network (ResNet) với 32 lớp, nhằm giải quyết các vấn đề như vanishing gradients khi huấn luyện các mạng rất sâu. Điều này giúp ResNet32 có thể học được các đặc trưng phức tạp của hình ảnh mà không bị giảm chất lượng thông tin.

```python
class ResNet32:
    def __init__(self, input_shape, num_classes, num_blocks=3):
        self.input_shape = input_shape
        self.num_classes = num_classes
        self.num_blocks = num_blocks  # Number of residual blocks per stage
        self.model = self.build_model()

    def resnet_block(self, x, filters, kernel_size=3, stride=1):
        shortcut = x
        if stride != 1 or shortcut.shape[-1] != filters:
            shortcut = tf.keras.layers.Conv2D(filters, 1, strides=stride, padding='same')(shortcut)
            shortcut = tf.keras.layers.BatchNormalization()(shortcut)

        x = tf.keras.layers.Conv2D(filters, kernel_size, strides=stride, padding='same')(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Activation('relu')(x)

        x = tf.keras.layers.Conv2D(filters, kernel_size, strides=1, padding='same')(x)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Add()([shortcut, x])
        x = tf.keras.layers.Activation('relu')(x)
        return x

    def build_model(self):
        inputs = tf.keras.Input(shape=self.input_shape)
        x = tf.keras.layers.Conv2D(16, 3, strides=1, padding='same')(inputs)
        x = tf.keras.layers.BatchNormalization()(x)
        x = tf.keras.layers.Activation('relu')(x)

        for _ in range(self.num_blocks):
            x = self.resnet_block(x, 16)
        for _ in range(self.num_blocks):
            x = self.resnet_block(x, 32, stride=2)
        for _ in range(self.num_blocks):
            x = self.resnet_block(x, 64, stride=2)

        x = tf.keras.layers.GlobalAveragePooling2D()(x)
        x = tf.keras.layers.Dropout(0.5)(x)
        outputs = tf.keras.layers.Dense(self.num_classes, activation='softmax')(x)

        model = tf.keras.Model(inputs, outputs)
        lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate=0.001, decay_steps=1000, decay_rate=0.9
        )
        model.compile(
            optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )
        return model
```

Hình 1: Mô hình ResNet32

2.3. Lý thuyết phòng thủ Defensive Distillation:

2.3.1. Khái niệm

- Defensive Distillation là một kỹ thuật được áp dụng để bảo vệ mô hình học máy, đặc biệt là các mô hình học sâu, khỏi các cuộc tấn công adversarial attacks (tấn công đối kháng).
- Phương pháp này dựa trên việc sử dụng distillation kết hợp với các soft labels thay vì các hard labels truyền thống để huấn luyện mô hình.

2.3.2. Quy trình

- Distillation:
    - Distillation là quá trình truyền đạt kiến thức từ một mô hình học sâu phức tạp (teacher model) sang một mô hình nhỏ hơn và đơn giản hơn (student model).
    - Mô hình teacher được huấn luyện trước với các nhãn thực tế (hard labels) và sau đó cung cấp soft labels cho mô hình student.
- Soft Labels:
    - Thay vì sử dụng nhãn chuẩn (hard labels như 0 hoặc 1 cho các lớp), soft labels là đầu ra xác suất từ mô hình teacher.
    - Ví dụ, thay vì nói rằng một hình ảnh là "chó" (hard label), soft labels cung cấp xác suất rằng hình ảnh đó có thể thuộc các lớp khác nhau, ví dụ: Chó (80%), Mèo (15%), Ngựa (5%).
- Defensive Distillation:
    - Defensive distillation sử dụng soft labels từ mô hình teacher để huấn luyện mô hình student.
    - Khi huấn luyện trên soft labels, mô hình học không chỉ học các đặc điểm chính xác mà còn học được những thông tin không chắc chắn, làm cho mô hình student trở nên robust (kháng cự) hơn với các perturbations nhỏ từ các cuộc tấn công adversarial.

2.3.3. Công thức:

- Softmax Output: Đầu ra của mô hình teacher là một vector xác suất, được tính bằng cách sử dụng nhiệt độ T trong hàm softmax.

$$p_i = \frac{e^{z_i/T}}{\sum_j e^{z_j/T}}$$

Trong đó:

  - p_i là xác suất của lớp i (soft label)
  - z_i là đầu ra của teacher model trước khi áp dụng softmax
  - T là nhiệt độ (temperature), điều chỉnh độ sắc nét của phân phối xác suất.

- Hàm mất mát trong Defensive Distillation thường sử dụng Cross-Entropy Loss giữa soft labels từ teacher và đầu ra của student model:

$$L = -\sum_i p_i log(q_i)$$

Trong đó:

  - p_i là soft label (xác suất từ teacher model)
  - q_i là xác suất dự đoán của student model (sau khi huấn luyện)

2.4. Lý thuyết phòng thủ Focal Loss & Hinge Loss

2.4.1. Khái niệm

Phương pháp phòng thủ kết hợp **Focal Loss** và **Hinge Loss** là một cách tiếp cận sáng tạo nhằm cải thiện hiệu suất của các mô hình học sâu trong các bài toán phân loại, đặc biệt là khi dữ liệu huấn luyện gặp phải sự mất cân bằng nhãn hoặc nhiễu. Sự kết hợp này tận dụng ưu điểm của cả hai hàm mất mát, giúp mô hình vừa tập trung vào các mẫu khó phân loại vừa duy trì được khoảng cách an toàn giữa các nhãn và biên quyết định.

- **Focal Loss**: là một phương pháp mở rộng từ Cross-Entropy Loss, được thiết kế để giải quyết các vấn đề liên quan đến mất cân bằng nhãn trong

dữ liệu. Đây là vấn đề phổ biến trong các bài toán mà số lượng mẫu thuộc một số lớp nhất định (ví dụ: các đối tượng hiếm gặp hoặc nhỏ) ít hơn đáng kể so với các lớp khác. Một ví dụ điển hình là phát hiện vật thể nhỏ hoặc các sự kiện hiếm trong dữ liệu video hoặc hình ảnh.

- **Ý tưởng chính** là giảm trọng số của các mẫu dễ phân loại (tức là các mẫu mà mô hình đã phân loại đúng với độ tin cậy cao), đồng thời tăng trọng số của các mẫu khó phân loại (những mẫu có xác suất phân loại đúng thấp). Bằng cách này, mô hình được "hướng dẫn" để tập trung vào việc học từ những mẫu khó khăn hơn, từ đó cải thiện khả năng của nó trên các tập dữ liệu bị nhiễu hoặc mất cân bằng. Focal Loss đạt được điều này bằng cách thêm một thành phần điều chỉnh trọng số dựa trên độ tin cậy của mô hình. Thành phần này làm giảm giá trị mất mát cho các mẫu dễ và tăng giá trị mất mát cho các mẫu khó, giúp mô hình phân bổ nguồn lực học tập hiệu quả hơn.
- **Hinge Loss:** là một hàm mất mát phổ biến trong các bài toán phân loại nhị phân, đặc biệt là trong các thuật toán như Support Vector Machine (SVM). Ý tưởng chính của Hinge Loss là tạo ra một **lề (margin)** an toàn giữa các lớp, tức là đẩy các điểm dữ liệu của từng lớp ra xa khỏi biên quyết định càng nhiều càng tốt. Biên quyết định này chính là ranh giới mà mô hình sử dụng để phân tách các lớp trong không gian đặc trưng.
- Hàm **Hinge Loss** được định nghĩa sao cho nếu một mẫu được phân loại chính xác và nằm cách xa biên quyết định đủ lớn (vượt quá một ngưỡng nhất định), thì hàm mất mát cho mẫu đó sẽ bằng 0. Tuy nhiên, nếu mẫu đó gần biên hoặc nằm sai bên của biên quyết định, giá trị mất mát sẽ tăng lên, buộc mô hình phải điều chỉnh để đảm bảo biên quyết định trở nên "sạch" và cách biệt rõ ràng giữa các lớp.

➢ Việc kết hợp hai hàm mất mát này mang lại lợi ích từ cả hai phương pháp. **Focal Loss** giúp mô hình tập trung vào các mẫu khó phân loại và

giảm thiểu ảnh hưởng của các mẫu dễ, điều này đặc biệt quan trọng trong dữ liệu không cân bằng hoặc bị nhiễu. Trong khi đó, **Hinge Loss** giúp đảm bảo rằng biên quyết định được mở rộng và tối ưu hóa, giữ cho các lớp được phân tách một cách an toàn.

2.4.2. Công thức

Công thức của Focal Loss:

$$\mathcal{L}_{focal}(y, \hat{y}) = -\alpha \,.\, (1 - \hat{y})^{\gamma} . y . \log(\hat{y})$$

ŷ: Xác suất dự đoán mô hình

α: Hệ số trọng số để cân bằng nhãn

y: Nhãn thực tế

γ: Hệ số tập trung, giá trị cao làm tăng trọng số cho mẫu khó

Công thức của Hinge Loss:

$$\mathcal{L}_{hinge}(y, \hat{y}) = max(0, 1 - y . \hat{y})$$

Công thức kết hợp của Focal Loss và Hinge Loss:

$$\mathcal{L}_{combination}(y, \hat{y}) = -\alpha \,.\, (1 - \hat{y})^{\gamma} . y . \log(\hat{y}) + max(0, m - y . \hat{y})$$

m: Biên quyết định (margin)

Thêm **m** giúp kiểm soát cường độ xử phạt với các mẫu không đáp ứng biên, đặc biệt hữu ích khi dữ liệu chứa nhiễu.

- Giúp mô hình phân biệt tốt giữa các lớp, đồng thời làm giảm ảnh hưởng của các sai sót nhẹ. Tham số **m** trong Hinge Loss giúp điều chỉnh mức độ ảnh hưởng của những mẫu sai lệch, trong khi **gamma** và **alpha** trong Focal Loss giúp tăng trọng số cho các mẫu khó phân loại.

2.5. Lý thuyết phòng thủ Gradient Masking

2.5.1. Khái niệm

Gradient Masking là một kỹ thuật được sử dụng trong lĩnh vực bảo mật học máy nhằm mục đích ngăn chặn hoặc làm phức tạp việc tính toán gradient của hàm mất mát

Mục tiêu chính của gradient masking là làm giảm khả năng của kẻ tấn công trong việc sử dụng gradient để tìm kiếm các ví dụ đầu vào gây nhầm lẫn (adversarial examples) hoặc khai thác lỗ hổng của mô hình.

2.5.2. Cơ chế hoạt động

Gradient masking hoạt động bằng cách làm biến đổi hàm mất mát hoặc kiến trúc của mô hình sao cho gradient không cung cấp thông tin hữu ích cho kẻ tấn công.

Một số phương pháp thực hiện Gradient Masking phổ biến:

- Sử Dụng Các Hàm Kích Hoạt Không Định Hình (Non-differentiable Activation Functions): Áp dụng các hàm kích hoạt như ReLU, Leaky ReLU, hoặc các hàm kích hoạt không liên tục có thể làm biến đổi gradient, khiến gradient trở nên không chính xác hoặc không ổn định.
- Thêm Nhiễu Vào Gradient:Thêm nhiễu ngẫu nhiên vào gradient trong quá trình huấn luyện hoặc suy luận để làm giảm độ chính xác của gradient.
- Sử Dụng Các Phương Pháp Tối Ưu Hạn Chế (Optimization Constraints): Áp đặt các ràng buộc hoặc giới hạn trên gradient để hạn chế khả năng kẻ tấn công tối ưu hóa các ví dụ adversarial.

3. CÀI ĐẶT CHƯƠNG TRÌNH

3.1. Grid Search

Tối ưu hóa tham số trong mô hình **ResNet32**:

```python
# Grid Search function
def grid_search(data_gen, x_train, y_train, x_test, y_test, param_grid):
    best_model = None
    best_metrics = None
    best_params = None
    results = []

    all_combinations = list(product(*param_grid.values()))
    param_names = list(param_grid.keys())

    for combination in all_combinations:
        params = dict(zip(param_names, combination))
        print(f"Testing with params: {params}")

        model = ResNet32((32, 32, 3), num_classes).model
        lr_schedule = tf.keras.optimizers.schedules.ExponentialDecay(
            initial_learning_rate=params["learning_rate"],
            decay_steps=1000,
            decay_rate=0.9
        )
        model.compile(
            optimizer=tf.keras.optimizers.Adam(learning_rate=lr_schedule),
            loss='categorical_crossentropy',
            metrics=['accuracy']
        )

        metrics = train_and_evaluate(
            model, data_gen, x_train, y_train, x_test, y_test,
            title=f"Grid Search (Params: {params})",
            batch_size=params["batch_size"],
            epochs=params["epochs"]
        )

        results.append((params, metrics))

        if best_metrics is None or metrics["accuracy"] > best_metrics["accuracy"]:
            best_model = model
            best_metrics = metrics
            best_params = params

    print(f"Best Parameters: {best_params}")
    print(f"Best Accuracy: {best_metrics['accuracy']}%")
    return best_model, best_metrics, best_params, results
```

Hình 2: Áp dụng Grid Search để tối ưu hóa tham số cho mô hình

```python
# Parameter grid
param_grid = {
    "learning_rate": [0.001, 0.0005, 0.0001],
    "batch_size": [32, 64, 128],
    "epochs": [5, 10]
}
```

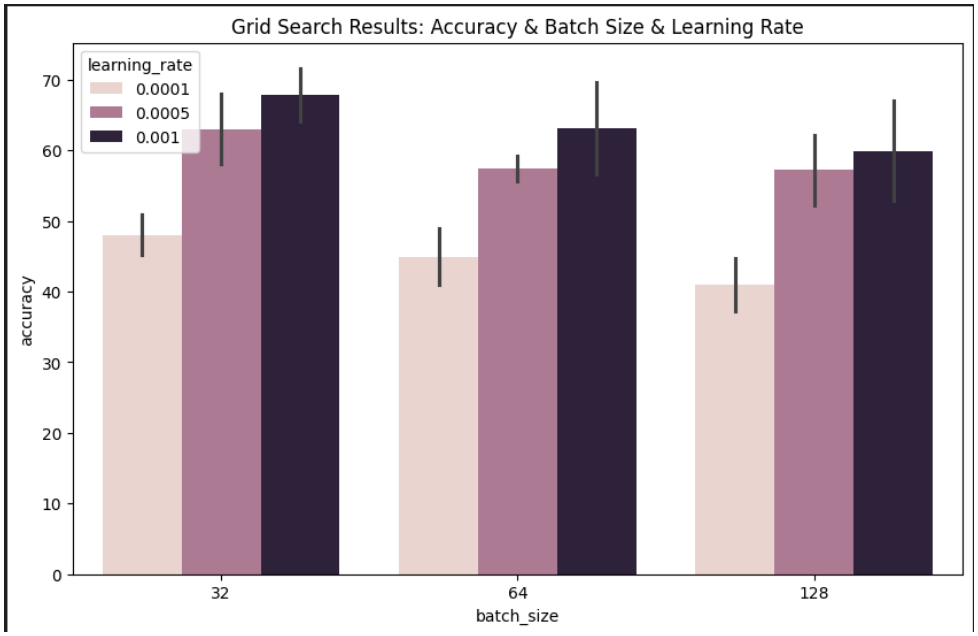
```python
# Run Grid Search
best_model, best_metrics, best_params, all_results = grid_search(
    datagen, x_train, y_train, x_test, y_test, param_grid
)
```

Hình 3: Thực hiện chạy các danh sách tham số trên để tối ưu hóa

	learning_rate	batch_size	epochs	accuracy	precision	recall	f1_score	\
1	0.0010	32	10	71.649998	0.715193	0.7165	0.705194	
3	0.0010	64	10	69.690001	0.702790	0.6969	0.688702	
7	0.0005	32	10	68.010002	0.683585	0.6801	0.670308	
5	0.0010	128	10	66.990000	0.689049	0.6699	0.661919	
0	0.0010	32	5	64.160001	0.647971	0.6416	0.636025	
11	0.0005	128	10	62.150002	0.617791	0.6215	0.604393	
9	0.0005	64	10	59.219998	0.604526	0.5922	0.573319	
6	0.0005	32	5	58.020002	0.582795	0.5802	0.559346	
2	0.0010	64	5	56.550002	0.585294	0.5655	0.539573	
8	0.0005	64	5	55.650002	0.551221	0.5565	0.542624	
4	0.0010	128	5	52.840000	0.565950	0.5284	0.511120	
10	0.0005	128	5	52.209997	0.544781	0.5221	0.511799	
13	0.0001	32	10	50.910002	0.504942	0.5091	0.494949	
15	0.0001	64	10	49.000001	0.484989	0.4900	0.480134	
12	0.0001	32	5	45.130000	0.451655	0.4513	0.440317	
17	0.0001	128	10	44.790000	0.445625	0.4479	0.436980	
14	0.0001	64	5	40.910000	0.399437	0.4091	0.391470	
16	0.0001	128	5	37.210000	0.374981	0.3721	0.352291	

	time
1	934.060839
3	832.145659
7	953.279225
5	790.717148
...	
12	499.693415
17	800.346067
14	463.803699
16	404.587314

Hình 4: Kết quả sau khi thực hiện grid search tối ưu hóa tham số



Hình 5: Trực quan hóa kết quả sau khi thực hiện grid search

- Sau khi thực hiện grid search để tối ưu hóa tham số learning_rate, batch_size, và epochs thì cho ra kết quả là danh sách các tham số (từ tốt nhất đến thấp nhất) khi áp dụng vào mô hình ResNet32. Bộ tham số cho ra kết quả cao nhất là **learning_rate: 0.001, batch_size: 32, epochs: 10**

3.2. Defensive Distillation

```python
class DefensiveDistillation:
    def __init__(self, teacher_model, x_train, y_train, temperature=5.0):
        self.teacher_model = teacher_model
        self.x_train = x_train
        self.y_train = y_train
        self.temperature = temperature

    def create_soft_targets(self):
        teacher_logits = self.teacher_model.predict(self.x_train)
        soft_targets = np.exp(teacher_logits / self.temperature) / np.sum(np.exp(teacher_logits / self.temperature), axis=1, keepdims=True)
        return soft_targets

    def train_student_model(self, student_model, batch_size=32, epochs=10, logger=None):
        soft_targets = self.create_soft_targets()
        # Nếu logger không phải None, thêm vào callback
        callbacks = [logger] if logger else []
        student_model.fit(
            self.x_train, soft_targets,
            batch_size=batch_size, epochs=epochs,
            callbacks=callbacks, verbose=1
        )
```

Hình 6: Xây dựng phòng thủ Defensive Distillation

```
data_handler = DataHandler()
x_train, y_train, x_test, y_test, num_classes = data_handler.load_and_preprocess_data()

datagen = tf.keras.preprocessing.image.ImageDataGenerator(
    rotation_range=15, width_shift_range=0.1, height_shift_range=0.1, horizontal_flip=True
)
datagen.fit(x_train)

teacher_results = []
teacher_logger = MetricsLogger(batch_size=32, results=teacher_results, x_val=x_test, y_val=y_test)
teacher_model = ResNet32((32, 32, 3), num_classes).model
teacher_model.fit(datagen.flow(x_train, y_train, batch_size=32), epochs=10, callbacks=[teacher_logger], verbose=1)

plot_results_table(teacher_results, "Teacher Model Results")
for metric in ['Accuracy', 'Precision', 'Recall', 'F1-score']:
    plot_individual_graph(teacher_results, metric, "Teacher Model Performance")
```

Hình 7: Giai đoạn một train Teacher Model (Original)

```
# Training Student Model
print("Training Student Model using Defensive Distillation...")
student_model = ResNet32((32, 32, 3), num_classes).model
student_results = []
student_logger = MetricsLogger(batch_size=32, results=student_results, x_val=x_test, y_val=y_test)
distillation = DefensiveDistillation(teacher_model, x_train, y_train)
distillation.train_student_model(student_model, batch_size=32, epochs=10, logger=student_logger)

plot_results_table(student_results, "Student Model Results")
for metric in ['Accuracy', 'Precision', 'Recall', 'F1-score']:
    plot_individual_graph(student_results, metric, "Student Model Performance")
```

Hình 8: Giai đoạn hai train Student Model

```
# Giai đoạn 3: Label Flipping Attack
print("Training Teacher Model with Label Flipping Attack...")
flipping_results = []
flipping_logger = MetricsLogger(batch_size=32, results=flipping_results, x_val=x_test, y_val=y_test)

# Tạo nhãn bị đảo
y_train_flipped = y_train.copy()
flip_indices = np.random.choice(len(y_train), int(0.5 * len(y_train)), replace=False)
for idx in flip_indices:
    true_label = np.argmax(y_train[idx])
    possible_labels = [i for i in range(num_classes) if i != true_label]
    flipped_label = np.random.choice(possible_labels)
    y_train_flipped[idx] = tf.keras.utils.to_categorical(flipped_label, num_classes)

# Huấn luyện mô hình với dữ liệu bị tấn công
teacher_model_flipped = ResNet32((32, 32, 3), num_classes).model
teacher_model_flipped.fit(
    datagen.flow(x_train, y_train_flipped, batch_size=32),
    epochs=10,
    callbacks=[flipping_logger],
    verbose=1
)
```

Hình 9: Giai đoạn ba thực hiện tấn công với Label Flipping Attack và train lại model

```python
def plot_comparison_graph(teacher_results, student_results, flipping_results):
    plt.figure(figsize=(10, 6))

    # Teacher Model Accuracy
    teacher_df = pd.DataFrame(teacher_results)
    if 'Accuracy' in teacher_df.columns:
        plt.plot(teacher_df['Epoch'], teacher_df['Accuracy'], label='Original', marker='o')

    # Student Model Accuracy
    student_df = pd.DataFrame(student_results)
    if 'Accuracy' in student_df.columns:
        plt.plot(student_df['Epoch'], student_df['Accuracy'], label='Original with Defensive Distillation', marker='o')

    # Flipping Attack Accuracy
    flipping_df = pd.DataFrame(flipping_results)
    if 'Accuracy' in flipping_df.columns:
        plt.plot(flipping_df['Epoch'], flipping_df['Accuracy'], label='Label Flipping Attack', marker='o')

    # Graph Formatting
    plt.xlabel('Epoch')
    plt.ylabel('Accuracy (%)')
    plt.title('Comparison of Accuracy Across Models')
    plt.legend()
    plt.grid(True)
    plt.show()
```

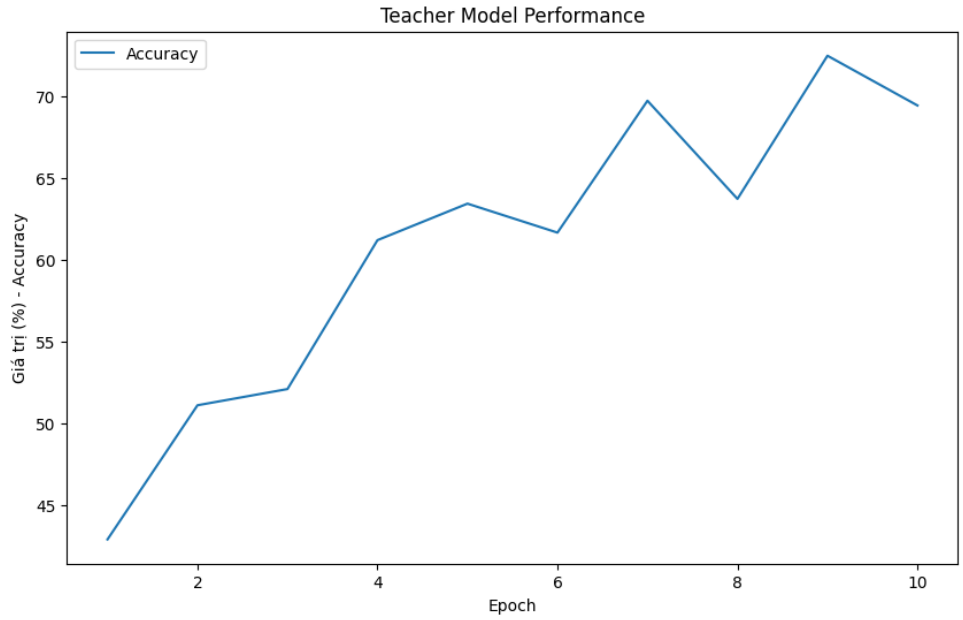
Hình 10: Trực quan hoá dữ liệu dạng bảng và dạng biểu đồ qua các giai đoạn

Trực quan hoá dữ liệu dạng bảng giai đoạn chạy các tham số trong Traning Teacher Model:

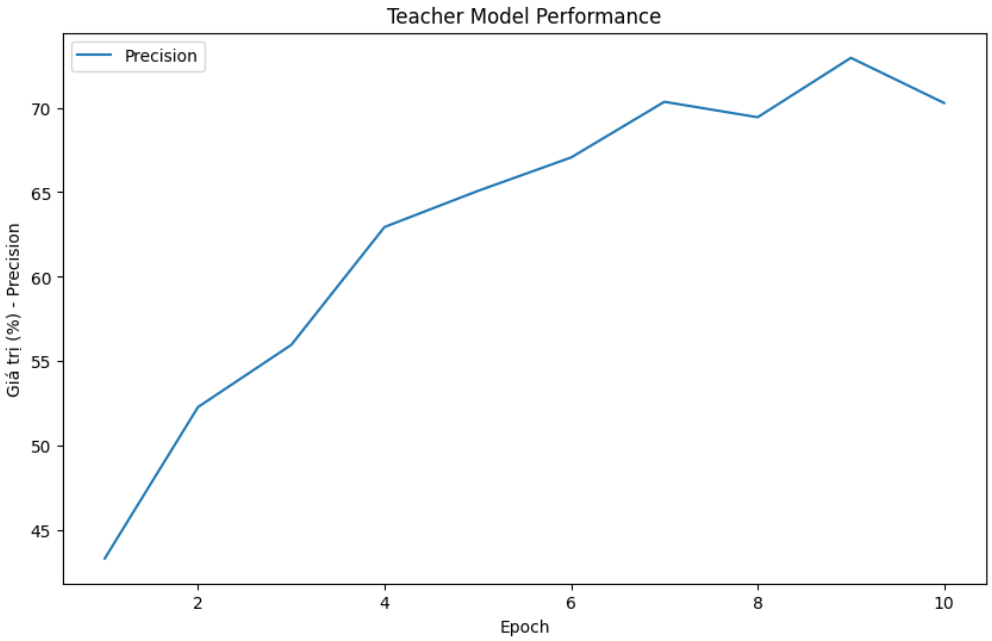
```
### Teacher Model Results ###
   Epoch  Batch Size  Accuracy  Precision  Recall   F1-score      Loss
0      1          32     42.89  43.343749   42.89  41.729819  1.899301
1      2          32     51.09  52.297708   51.09  50.083242  1.546821
2      3          32     52.08  55.974425   52.08  49.453048  1.374396
3      4          32     61.18  62.945013   61.18  59.426944  1.217963
4      5          32     63.41  65.076023   63.41  62.552199  1.111433
5      6          32     61.64  67.054612   61.64  60.119242  1.044519
6      7          32     69.70  70.347906   69.70  68.554928  0.987048
7      8          32     63.70  69.434598   63.70  61.986268  0.944109
8      9          32     72.45  72.946660   72.45  72.043097  0.900331
9     10          32     69.41  70.276097   69.41  68.105962  0.871989
```

Hình 11: Bảng chạy các tham số mô hình học giai đoạn Training Teacher Model

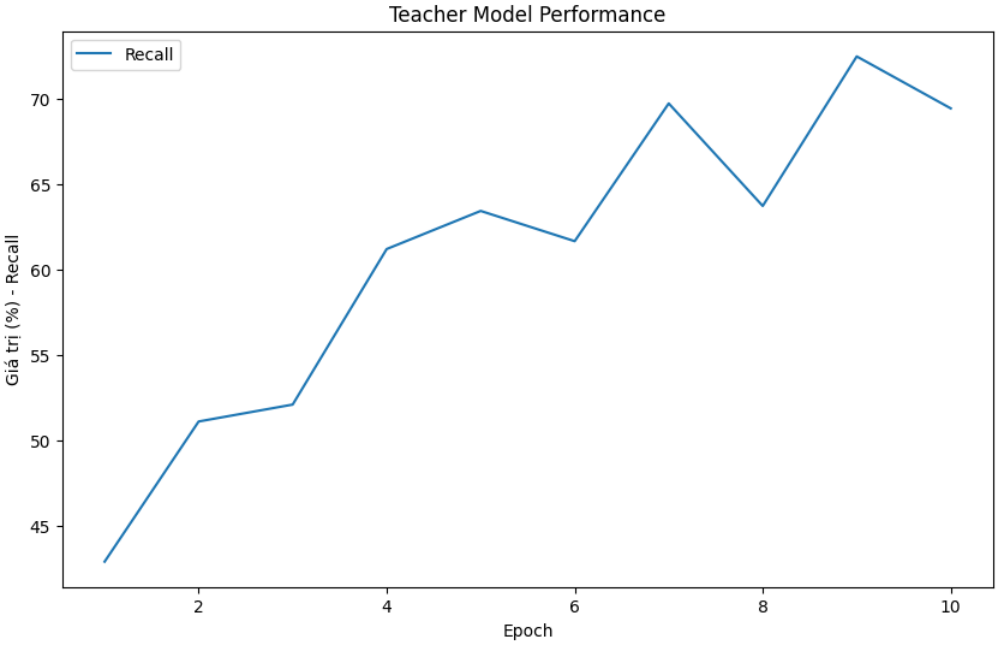
Trực quan dữ liệu dạng biểu đồ Recall, F1-score, Accuracy, Precision trên epoch giai đoạn Training Teacher Model:



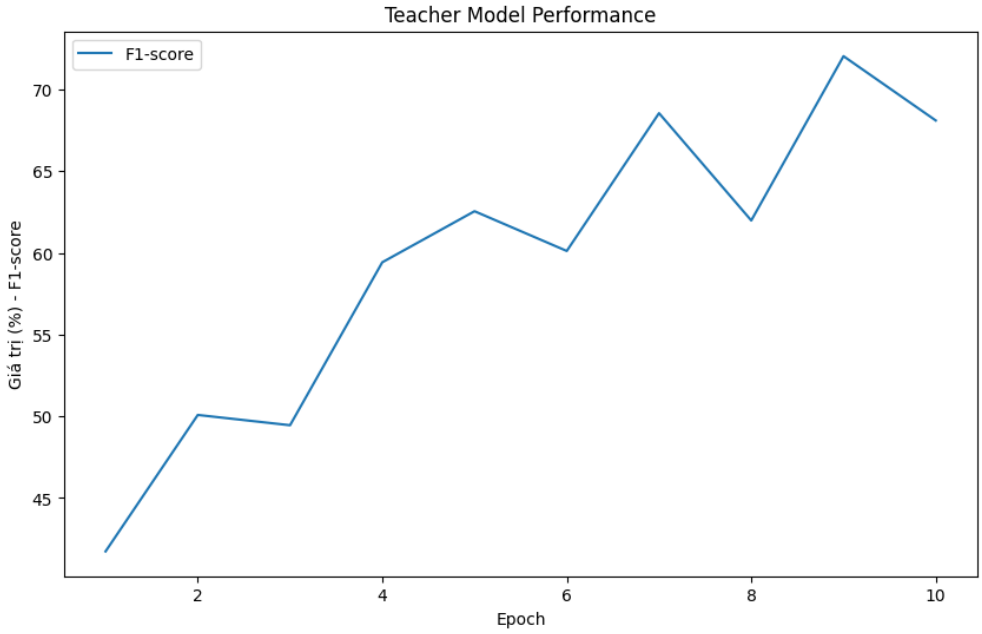
Hình 12: Biểu đồ Accuary trên epoch giai đoạn Teacher Model



Hình 13: Biểu đồ Precision trên epoch giai đoạn Teacher Model



Hình 14: Biểu đồ Recall trên epoch giai đoạn Teacher Model



Hình 15: Biểu đồ F1-score trên epoch giai đoạn Teacher Model

Diễn giải các biểu đồ trên:

- Ban đầu (Epoch 1 - 2):
  - Mức độ chính xác của mô hình ở mức thấp.
  - Đây là giai đoạn đầu của quá trình huấn luyện, khi mô hình đang học từ dữ liệu.

- Tăng trưởng đều (Epoch 2 - 5):
    - Độ chính xác tăng đáng kể, đạt hơn vào epoch thứ 5.
    - Điều này thể hiện mô hình học được các đặc trưng quan trọng từ dữ liệu huấn luyện.
- Biến động (Epoch 5 - 10):
    - Độ chính xác dao động nhưng có xu hướng tăng chung:
    - Giảm nhẹ từ epoch 6 xuống epoch 7.
    - Tăng mạnh ở epoch 8, đạt mức cao nhất
- Giảm nhẹ ở epoch 10.
    - Giai đoạn này thường xảy ra khi mô hình đã gần đạt ngưỡng học tối đa hoặc đang tiếp cận hiện tượng overfitting.

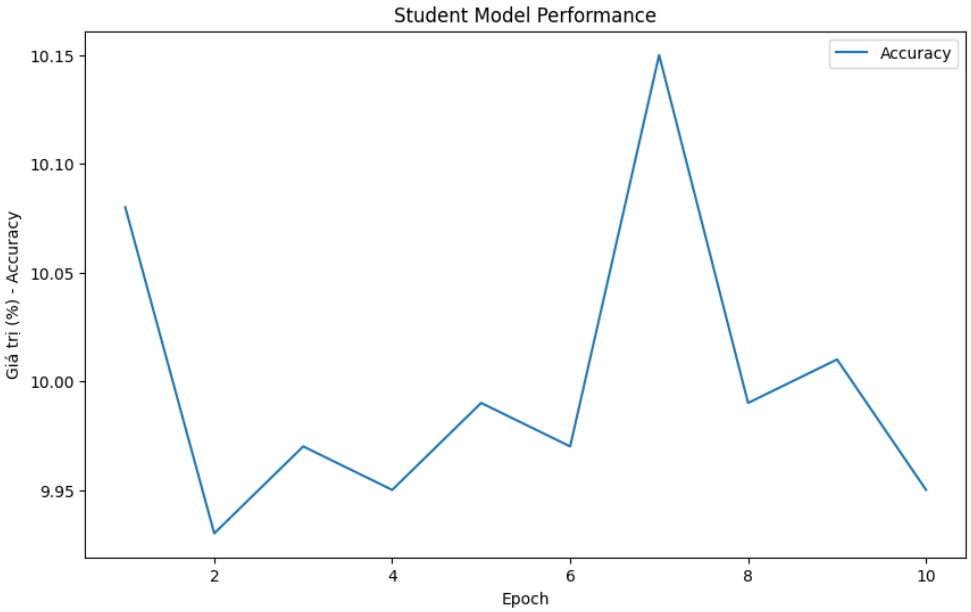
Trực quan hoá dữ liệu dạng bảng giai đoạn chạy các tham số trong Traning Student Model:

```
### Student Model Results ###
c:\Users\Windows\AppData\Local\Programs\Python\Python312\Lib\site-packag
  _warn_prf(average, modifier, f"{metric.capitalize()} is", len(result))
```

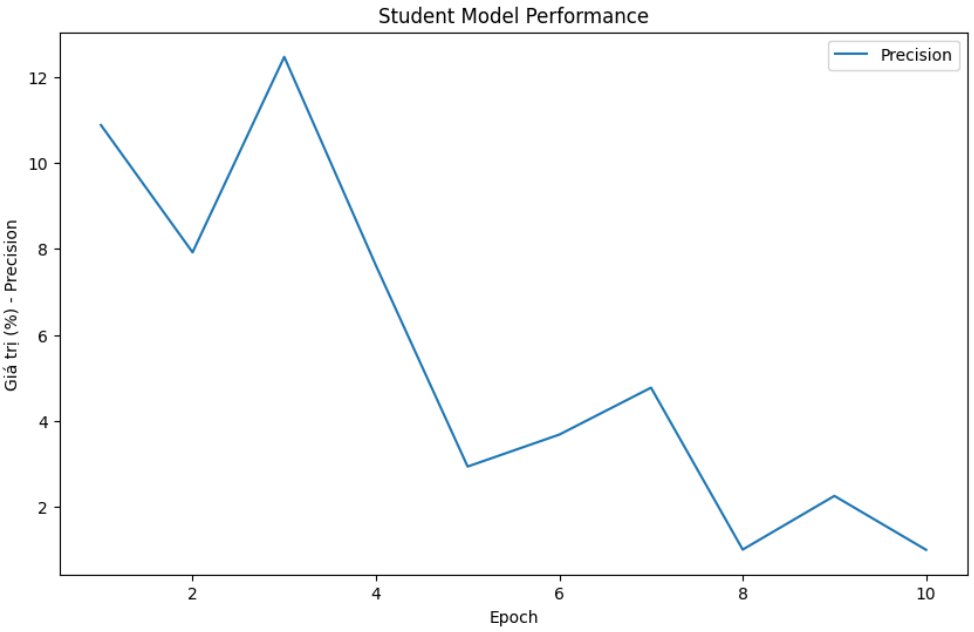
	Epoch	Batch Size	Accuracy	Precision	Recall	F1-score	Loss
0	1	32	10.08	10.881313	10.08	3.850340	2.331077
1	2	32	9.93	7.918540	9.93	2.290540	2.302568
2	3	32	9.97	12.466593	9.97	2.064837	2.302549
3	4	32	9.95	7.609779	9.95	1.928143	2.302547
4	5	32	9.99	2.934176	9.99	1.931251	2.302547
5	6	32	9.97	3.677467	9.97	1.855230	2.302548
6	7	32	10.15	4.768953	10.15	2.282069	2.302546
7	8	32	9.99	1.002006	9.99	1.821331	2.302548
8	9	32	10.01	2.250701	10.01	1.858545	2.302546
9	10	32	9.95	0.996295	9.95	1.811231	2.302543

Hình 16: Bảng chạy các tham số mô hình học giai đoạn Training Student Model

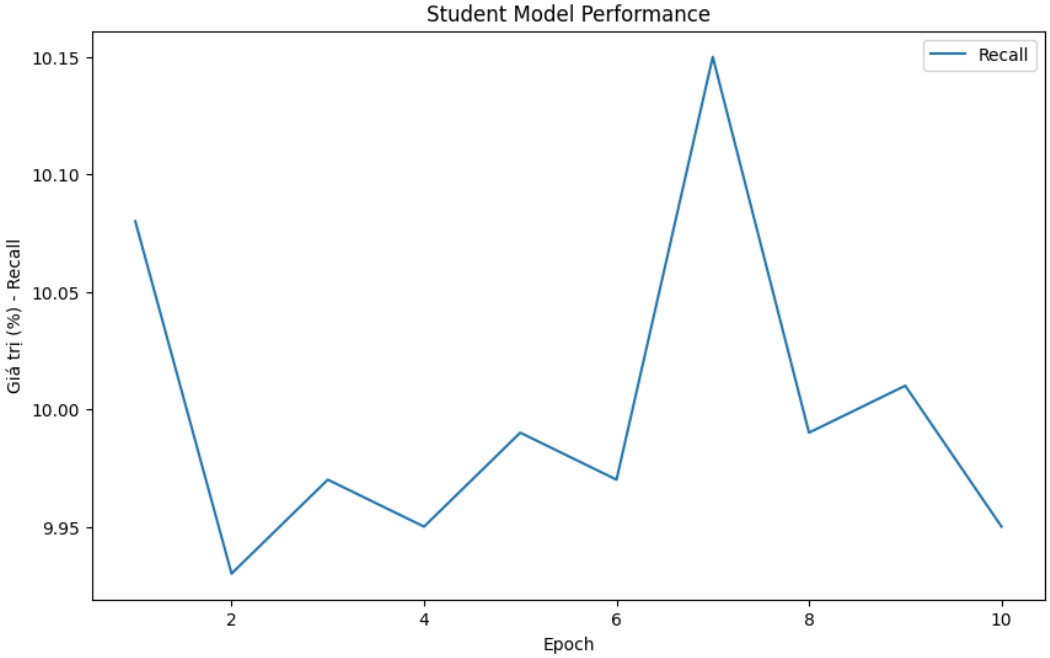
Trực quan dữ liệu dạng biểu đồ Recall, F1-score, Accuracy, Precision trên epoch giai đoạn Training Student Model:



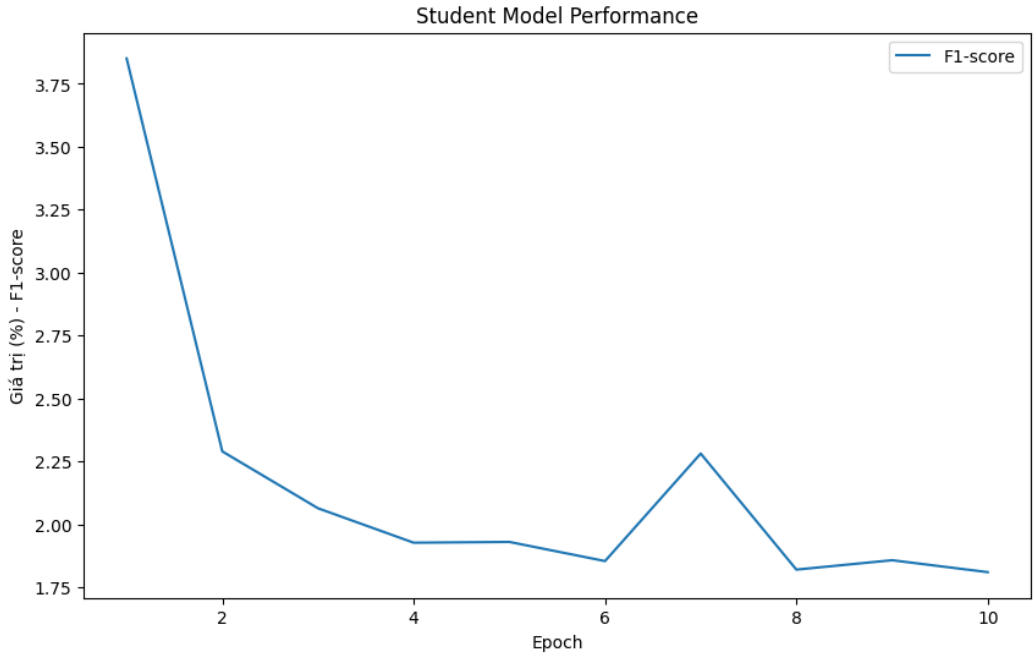
Hình 17: Biểu đồ Accuracy trên epoch giai đoạn Student Model



Hình 18: Biểu đồ Precision trên epoch giai đoạn Student Model



Hình 19: Biểu đồ Recall trên epoch giai đoạn Student Model



Hình 20: Biểu đồ F1-score trên epoch giai đoạn Student Model

Diễn giải các biểu đồ trên:

- Accuracy, Recall

- Biểu đồ dao động và có sự giảm nhẹ từ các epoch ban đầu (~10.1%) xuống thấp hơn (~9.9%) trước khi tăng đột ngột tại epoch 6 (~10.15%). Sau đó, giá trị giảm mạnh và tiếp tục dao động nhẹ ở các epoch cuối.
- Điều này cho thấy quá trình huấn luyện có sự bất ổn định ở một số giai đoạn. Nguyên nhân có thể là do mô hình đang cố gắng điều chỉnh để học từ các soft labels được tạo từ Defensive Distillation.

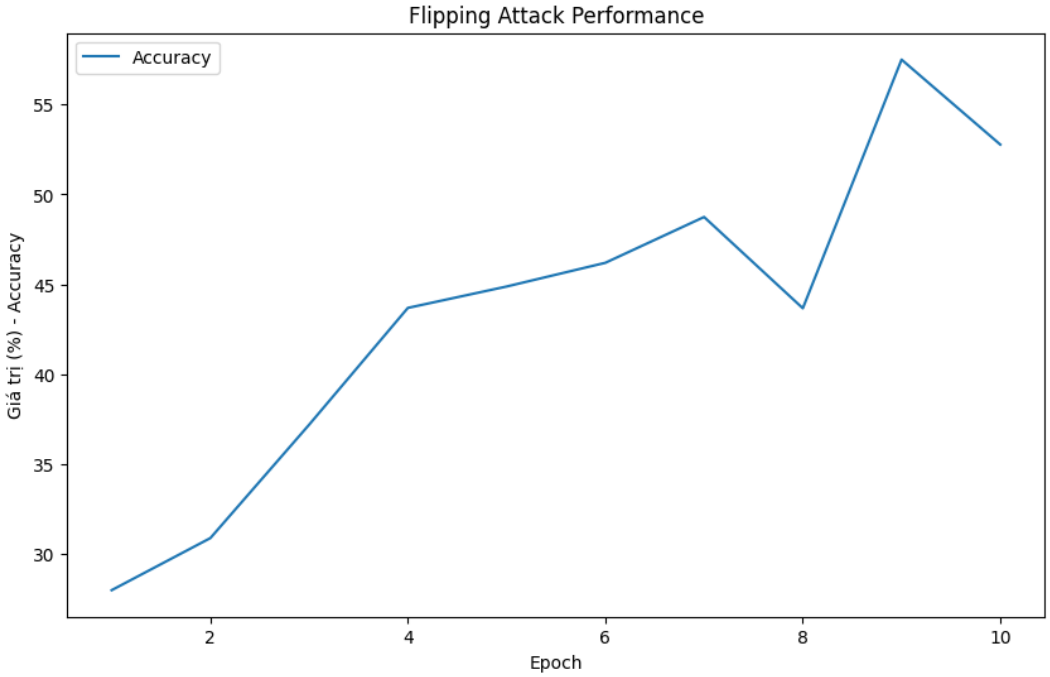
- Precision, F1-score
  - Precision bắt đầu ở mức khá cao (~12%) tại epoch đầu tiên, sau đó giảm mạnh (~4-6% ở giữa) và tiếp tục giảm xuống gần như không đáng kể (~2%) ở epoch cuối.
  - Điều này cho thấy mô hình Student có vẻ không tối ưu khi xác định các dự đoán đúng so với các nhãn thực tế, đặc biệt khi xử lý dữ liệu được tạo ra từ Distillation.

Trực quan hoá dữ liệu dạng bảng giai đoạn chạy các tham số giai đoạn Label Flipping Attack:

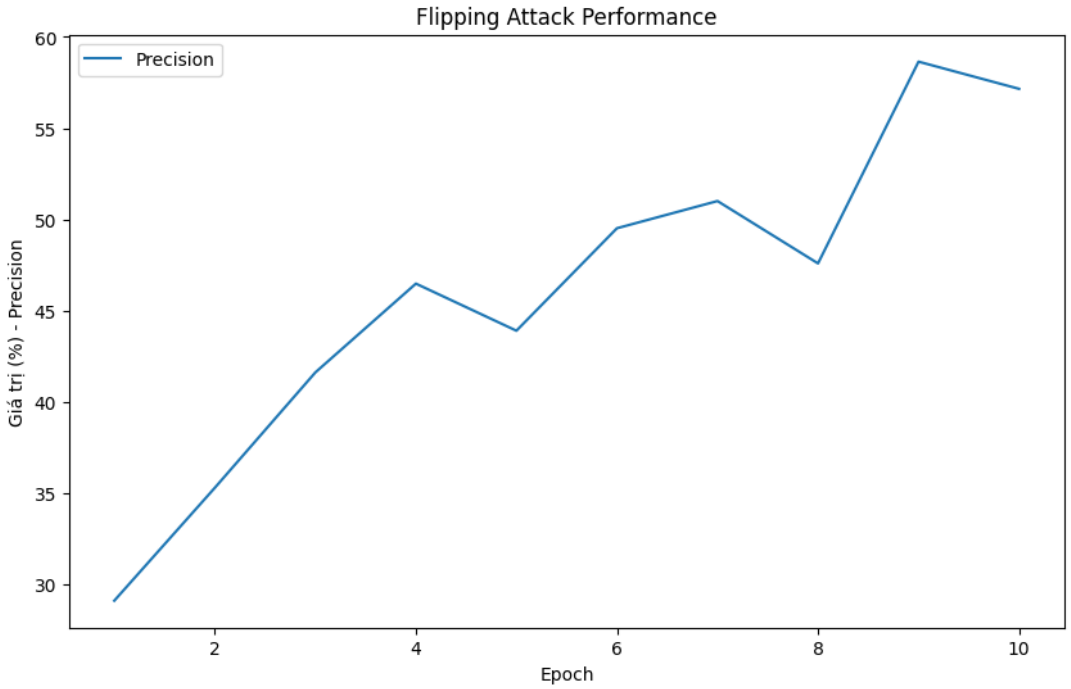
```
### Flipping Attack Results ###
   Epoch  Batch Size  Accuracy  Precision  Recall   F1-score      Loss
0      1          32     27.97  29.100092   27.97  24.862637  2.332254
1      2          32     30.87  35.287888   30.87  28.168542  2.236832
2      3          32     37.18  41.611207   37.18  34.342858  2.209732
3      4          32     43.68  46.477317   43.68  42.179019  2.191328
4      5          32     44.87  43.896852   44.87  40.778159  2.174264
5      6          32     46.19  49.512515   46.19  43.639453  2.159086
6      7          32     48.74  51.004325   48.74  46.394402  2.143927
7      8          32     43.66  47.585055   43.66  39.821359  2.133170
8      9          32     57.50  58.643265   57.50  55.926677  2.123219
9     10          32     52.77  57.153353   52.77  51.132305  2.112979
```

Hình 21: Bảng chạy các tham số mô hình học giai đoạn tấn công Label Flipping Attack

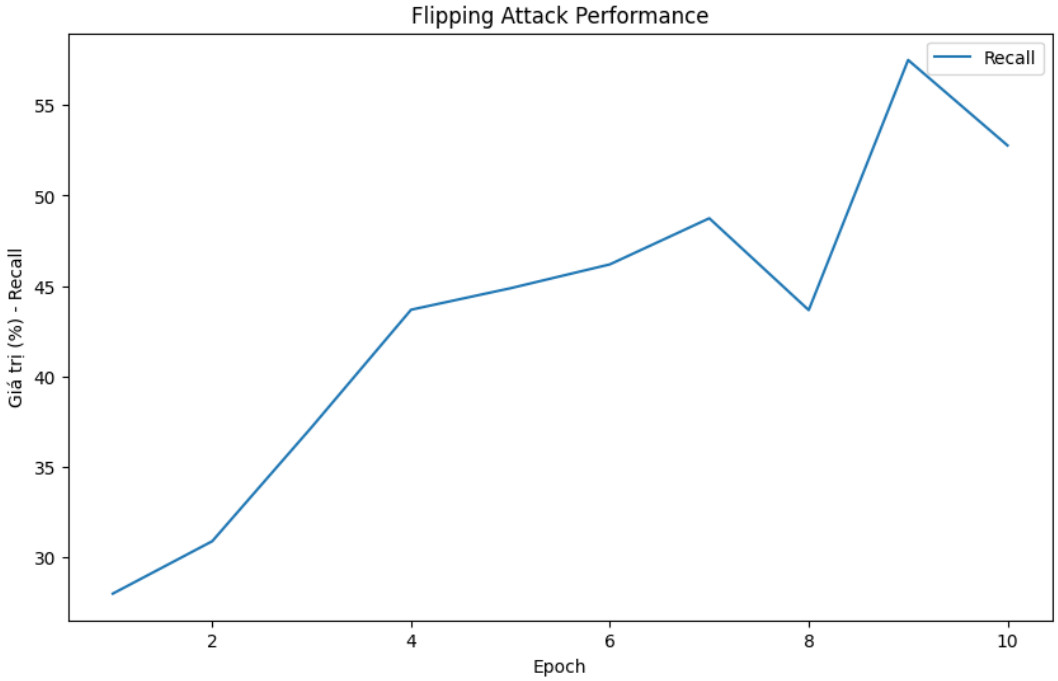
Trực quan dữ liệu dạng biểu đồ Recall, F1-score, Accuracy, Precision trên epoch giai đoạn Label Flipping Attack:



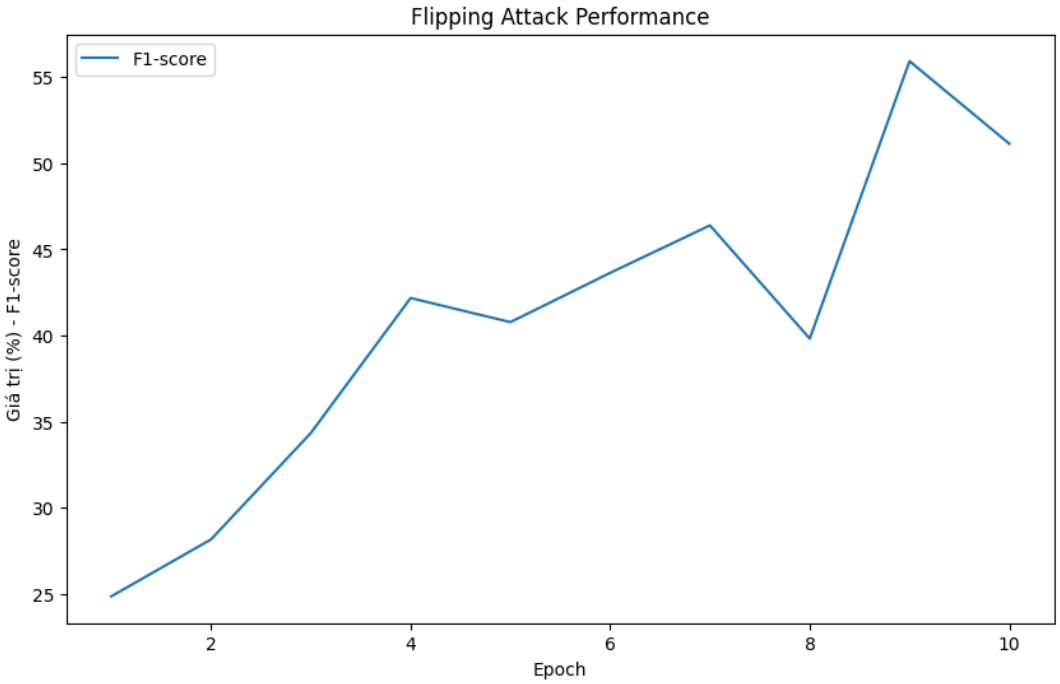
Hình 22: Biểu đồ Accuracy trên epoch giai đoạn Label Flipping Attack



Hình 23: Biểu đồ Precision trên epoch giai đoạn Label Flipping Attack



Hình 24: Biểu đồ Recall trên epoch giai đoạn Label Flipping Attack



Hình 25: Biểu đồ F1-score trên epoch giai đoạn Label Flipping Attack

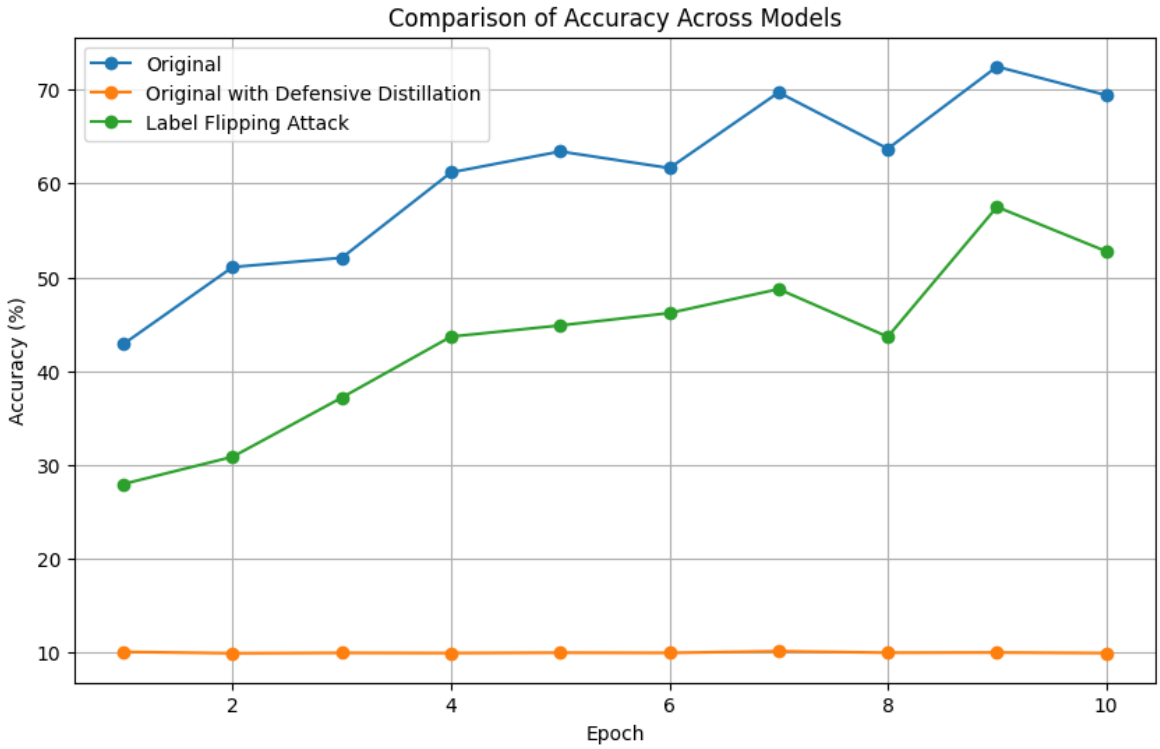
Diễn giải các biểu đồ trên:

- Tổng quan:

- Độ chính xác bắt đầu từ khoảng 30% tại epoch 1 và có xu hướng tăng dần qua các epoch. Tuy nhiên, xu hướng tăng không đều, mà có sự dao động đáng kể trong một số epoch.
- Từ epoch 1 đến 5: Độ chính xác tăng đều đặn, từ khoảng 30% lên hơn 45%. Điều này cho thấy mô hình vẫn đang học, bất chấp dữ liệu đã bị tấn công bằng việc đảo nhãn (label flipping).
- Từ epoch 6 đến 8: Hiệu suất ổn định, nhưng tăng chậm hơn so với giai đoạn đầu.
- Từ epoch 8 đến 10: Độ chính xác tăng đột biến và đạt đỉnh ở khoảng 57%, nhưng sau đó lại giảm nhẹ. Điều này có thể do mô hình bắt đầu overfit hoặc các mẫu dữ liệu đảo nhãn gây nhiễu.

- Kết luận:
  - Ảnh hưởng của tấn công: So với một mô hình không bị tấn công, độ chính xác của mô hình bị giảm sút nghiêm trọng (chỉ đạt tối đa 57%). Điều này chứng tỏ Label Flipping Attack đã ảnh hưởng đến khả năng học và tổng quát hóa của mô hình.
  - Hiệu suất học: Mặc dù bị tấn công, mô hình vẫn có thể học được các mẫu từ dữ liệu, nhưng tốc độ và độ chính xác bị ảnh hưởng rõ rệt.

Biểu đồ đường so sánh Accuracy cả 3 giai đoạn



Hình 26: Biểu đồ đường so sánh Accuracy cả 3 giai đoạn

Diễn giải biểu đồ:

- Mô hình gốc (Original/Teacher Model):
  - Đường màu xanh biểu diễn độ chính xác tăng dần qua các epoch và đạt khoảng 70% ở epoch cuối cùng.
  - Điều này cho thấy mô hình gốc có khả năng học tốt từ dữ liệu không bị xáo trộn.
- Mô hình với Defensive Distillation (Màu cam):
  - Độ chính xác hầu như không tăng qua các epoch, dao động ở mức 10%.
  - Điều này chứng tỏ rằng kỹ thuật chưng cất phòng thủ không hiệu quả trong trường hợp này hoặc có thể chưa được triển khai đúng cách trong code.
- Mô hình dưới tấn công đảo nhãn (Label Flipping Attack - Màu xanh lá):

- Đường biểu diễn cho thấy độ chính xác bắt đầu ở mức thấp (~30%) nhưng cải thiện theo thời gian, đạt khoảng 50% ở cuối quá trình huấn luyện.
- Điều này cho thấy ngay cả khi nhãn bị xáo trộn, mô hình vẫn học được một số đặc trưng từ dữ liệu.

Kết luận:

- Mô hình gốc hoạt động tốt nhất, với độ chính xác cao nhất (~70%).
- Mô hình với Defensive Distillation không hiệu quả, khả năng phòng thủ không phát huy tác dụng trong bối cảnh này.
- Mô hình bị tấn công đảo nhãn có độ chính xác thấp hơn (~50%), nhưng vẫn hoạt động tốt hơn so với kỹ thuật phòng thủ.

3.3. Gradient Masking

```
# ----------------------------#
#          Training           #
# ----------------------------#

# Create the original ResNet32 model
input_shape = x_train.shape[1:]
model_original = ResNet32(input_shape=input_shape, num_classes=num_classes).model

# Train the original model
print("\nTraining Original ResNet32 Model...")
history_original = model_original.fit(
    x_train, y_train_cat,
    epochs=10,  # Adjust epochs as needed
    batch_size=64,  # Adjust batch size as needed
    validation_data=(x_val, y_val_cat),
    verbose=2
)

# Evaluate the original model on clean test data
test_loss, test_acc = model_original.evaluate(x_test, y_test_cat, verbose=0)
print(f"\nTest Accuracy before attack: {test_acc:.4f}")
```

Hình 27: Training Model

```
Training Original ResNet32 Model...
Epoch 1/10
665/665 - 435s - 654ms/step - accuracy: 0.2962 - loss: 1.9352 - val_accuracy: 0.3948 - val_loss: 1.6601
Epoch 2/10
665/665 - 431s - 649ms/step - accuracy: 0.4612 - loss: 1.4936 - val_accuracy: 0.4536 - val_loss: 1.5220
Epoch 3/10
665/665 - 432s - 650ms/step - accuracy: 0.5404 - loss: 1.3044 - val_accuracy: 0.5340 - val_loss: 1.3139
Epoch 4/10
665/665 - 500s - 752ms/step - accuracy: 0.5922 - loss: 1.1726 - val_accuracy: 0.5568 - val_loss: 1.2943
Epoch 5/10
665/665 - 451s - 678ms/step - accuracy: 0.6341 - loss: 1.0568 - val_accuracy: 0.5801 - val_loss: 1.2068
Epoch 6/10
665/665 - 452s - 680ms/step - accuracy: 0.6697 - loss: 0.9649 - val_accuracy: 0.5707 - val_loss: 1.2495
Epoch 7/10
665/665 - 419s - 630ms/step - accuracy: 0.7018 - loss: 0.8730 - val_accuracy: 0.6128 - val_loss: 1.1568
Epoch 8/10
665/665 - 408s - 614ms/step - accuracy: 0.7293 - loss: 0.7947 - val_accuracy: 0.6165 - val_loss: 1.2061
Epoch 9/10
665/665 - 441s - 663ms/step - accuracy: 0.7568 - loss: 0.7129 - val_accuracy: 0.6321 - val_loss: 1.1687
Epoch 10/10
665/665 - 439s - 660ms/step - accuracy: 0.7823 - loss: 0.6409 - val_accuracy: 0.6521 - val_loss: 1.1505

Test Accuracy before attack: 0.6531
```

Hình 28: Độ chính xác trước khi tấn công là 65%

```
# Create a new ResNet32 model for poisoned data
model_poisoned = ResNet32(input_shape=input_shape, num_classes=num_classes).model

# Train the poisoned model
print("\nTraining Poisoned ResNet32 Model with Label Flipping...")
history_poisoned = model_poisoned.fit(
    x_train, y_train_poisoned_cat,
    epochs=10,
    batch_size=64,
    validation_data=(x_val, y_val_cat),
    verbose=2
)

# Evaluate the poisoned model on clean test data
test_loss_poisoned, test_acc_poisoned = model_poisoned.evaluate(x_test, y_test_cat, verbose=0)
print(f"\nTest Accuracy after attack on clean data: {test_acc_poisoned:.4f}")
```

Hình 29: Thực hiện tấn công model

```
Training Poisoned ResNet32 Model with Label Flipping...
Epoch 1/10
665/665 - 487s - 732ms/step - accuracy: 0.2209 - loss: 2.1940 - val_accuracy: 0.3799 - val_loss: 1.7493
Epoch 2/10
665/665 - 446s - 671ms/step - accuracy: 0.3373 - loss: 1.9396 - val_accuracy: 0.4391 - val_loss: 1.5805
Epoch 3/10
665/665 - 415s - 624ms/step - accuracy: 0.3856 - loss: 1.8447 - val_accuracy: 0.4605 - val_loss: 1.5536
Epoch 4/10
665/665 - 443s - 666ms/step - accuracy: 0.4212 - loss: 1.7837 - val_accuracy: 0.4244 - val_loss: 1.6641
Epoch 5/10
665/665 - 440s - 662ms/step - accuracy: 0.4611 - loss: 1.7104 - val_accuracy: 0.4803 - val_loss: 1.5167
Epoch 6/10
665/665 - 405s - 608ms/step - accuracy: 0.4943 - loss: 1.6461 - val_accuracy: 0.5296 - val_loss: 1.3870
Epoch 7/10
665/665 - 435s - 654ms/step - accuracy: 0.5204 - loss: 1.5903 - val_accuracy: 0.4985 - val_loss: 1.4716
Epoch 8/10
665/665 - 410s - 617ms/step - accuracy: 0.5460 - loss: 1.5337 - val_accuracy: 0.5665 - val_loss: 1.3045
Epoch 9/10
665/665 - 458s - 689ms/step - accuracy: 0.5705 - loss: 1.4707 - val_accuracy: 0.4796 - val_loss: 1.5325
Epoch 10/10
665/665 - 425s - 639ms/step - accuracy: 0.5937 - loss: 1.4055 - val_accuracy: 0.5112 - val_loss: 1.4515

Test Accuracy after attack on clean data: 0.5059
```

Hình 30: Độ chính xác của mô hình khi bị tấn công giảm còn 50 %

```python
# ------------------------------#
#          Defended Model       #
# ------------------------------#
# Create the defended ResNet32 model
model_defended = DefendedResNet32(input_shape=input_shape, num_classes=num_classes).model

# Train the defended model on poisoned data
print("\nTraining Defended ResNet32 Model with Label Flipping...")
history_defended = model_defended.fit(
    x_train, y_train_poisoned_cat,
    epochs=10,
    batch_size=64,
    validation_data=(x_val, y_val_cat),
    verbose=2
)

# Evaluate the defended model on clean test data
test_loss_defended, test_acc_defended = model_defended.evaluate(x_test, y_test_cat, verbose=0)
print(f"\nTest Accuracy of defended model on clean data: {test_acc_defended:.4f}")
```

Hình 31: Xây dựng mô hình phòng thủ

```
Training Defended ResNet32 Model with Label Flipping...
Epoch 1/10
665/665 - 440s - 661ms/step - accuracy: 0.2230 - loss: 2.1971 - val_accuracy: 0.3520 - val_loss: 1.7944
Epoch 2/10
665/665 - 430s - 647ms/step - accuracy: 0.3385 - loss: 1.9334 - val_accuracy: 0.4197 - val_loss: 1.6479
Epoch 3/10
665/665 - 410s - 617ms/step - accuracy: 0.3926 - loss: 1.8366 - val_accuracy: 0.4300 - val_loss: 1.6453
Epoch 4/10
665/665 - 408s - 613ms/step - accuracy: 0.4336 - loss: 1.7586 - val_accuracy: 0.3868 - val_loss: 1.7415
Epoch 5/10
665/665 - 439s - 660ms/step - accuracy: 0.4722 - loss: 1.6900 - val_accuracy: 0.5279 - val_loss: 1.3920
Epoch 6/10
665/665 - 468s - 704ms/step - accuracy: 0.5007 - loss: 1.6243 - val_accuracy: 0.4876 - val_loss: 1.5074
Epoch 7/10
665/665 - 420s - 631ms/step - accuracy: 0.5307 - loss: 1.5623 - val_accuracy: 0.5467 - val_loss: 1.3569
Epoch 8/10
665/665 - 411s - 619ms/step - accuracy: 0.5528 - loss: 1.5044 - val_accuracy: 0.5495 - val_loss: 1.3546
Epoch 9/10
665/665 - 436s - 656ms/step - accuracy: 0.5793 - loss: 1.4434 - val_accuracy: 0.5852 - val_loss: 1.2515
Epoch 10/10
665/665 - 438s - 659ms/step - accuracy: 0.6015 - loss: 1.3914 - val_accuracy: 0.5653 - val_loss: 1.3081

Test Accuracy of defended model on clean data: 0.5715
```

Hình 32: Kết quả Accuracy tăng lên 57%

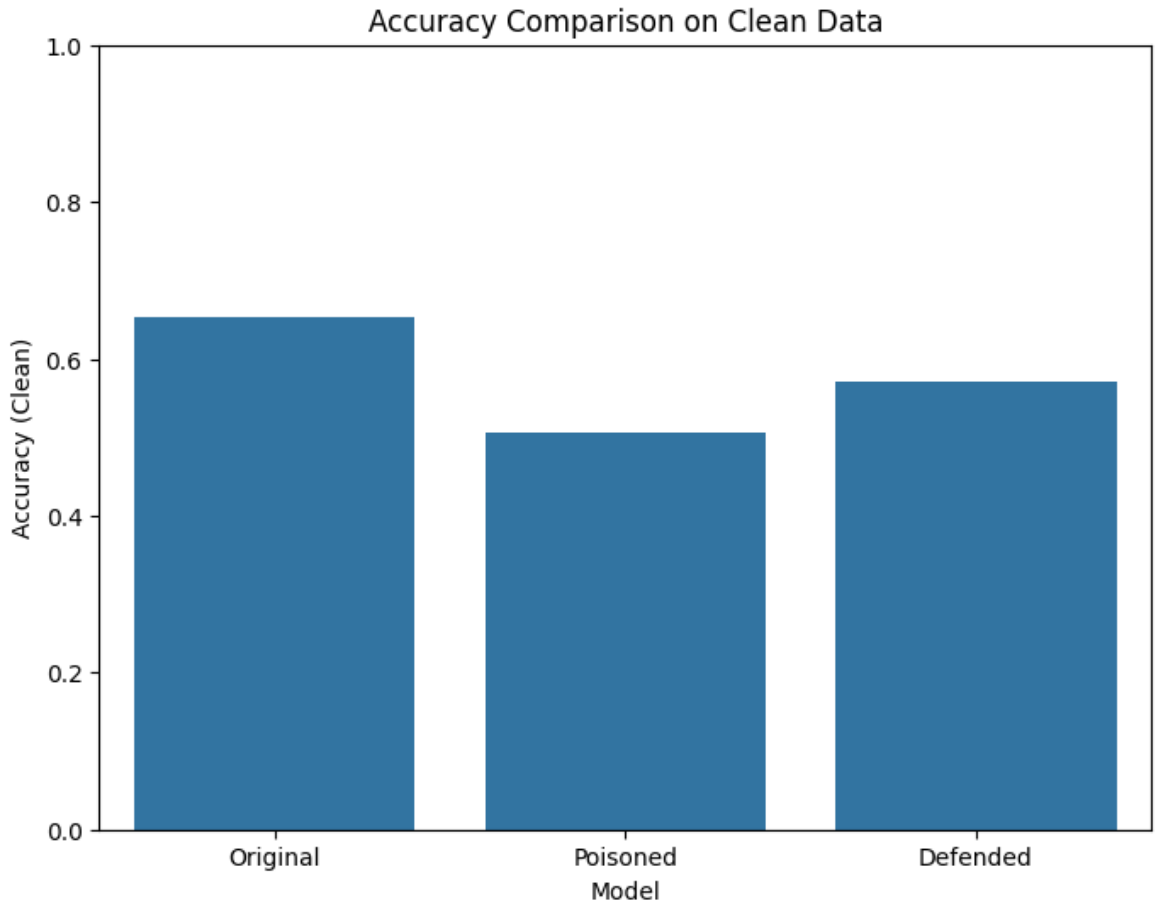
```
Metrics Comparison:
      Model  Accuracy (Clean)  Precision (Clean)  Recall (Clean)  \
0  Original            0.6531           0.660563          0.6531
1  Poisoned            0.5059           0.538723          0.5059
2  Defended            0.5715           0.581554          0.5715

   F1-Score (Clean)
0          0.653159
1          0.508614
2          0.571553
```

Hình 33: Metrics Comparison

Model Performance Metrics

Model	Accuracy (Clean)	Precision (Clean)	Recall (Clean)	F1-Score (Clean)
Original	0.6531	0.6606	0.6531	0.6532
Poisoned	0.5059	0.5387	0.5059	0.5086
Defended	0.5715	0.5816	0.5715	0.5716

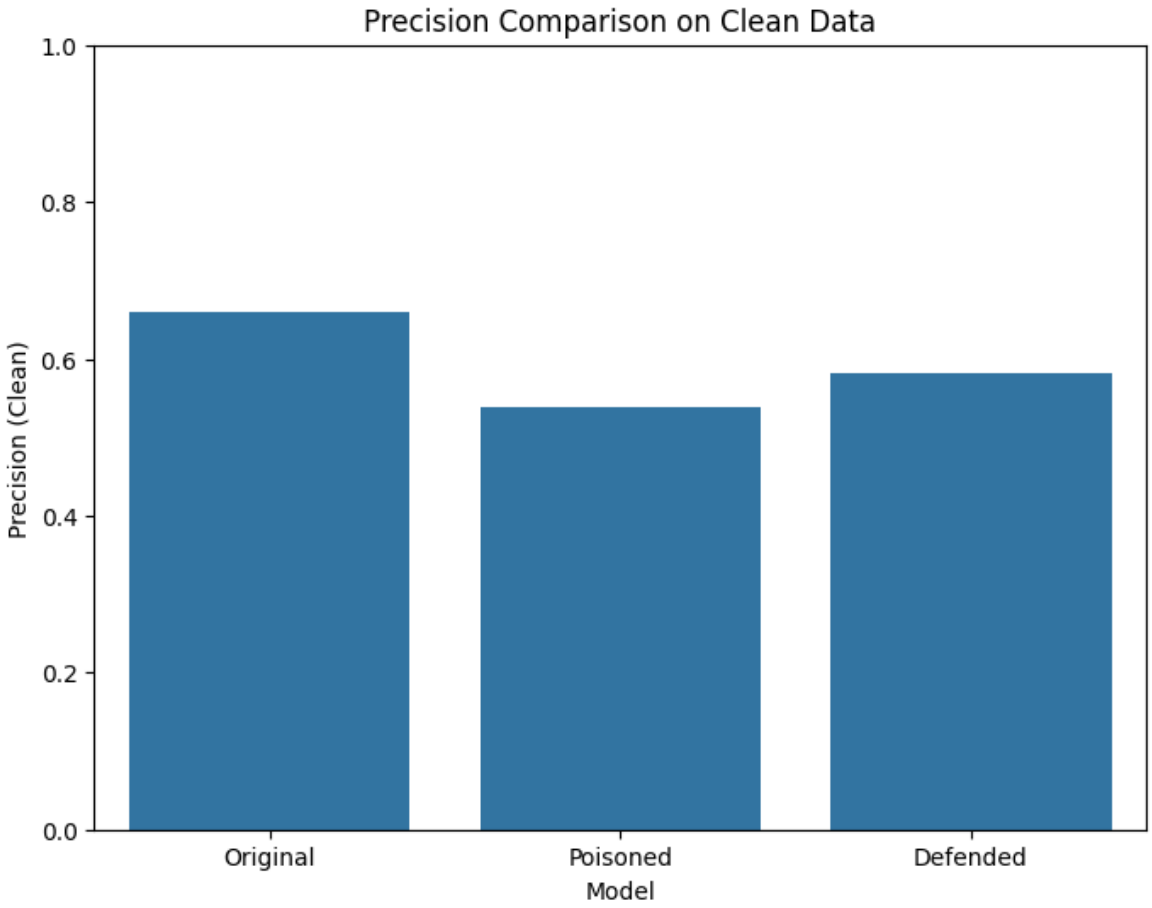
Hình 34: Metrics Comparison dạng bảng



Hình 35: Biểu đồ Accuracy

❖ **Nhận xét:**

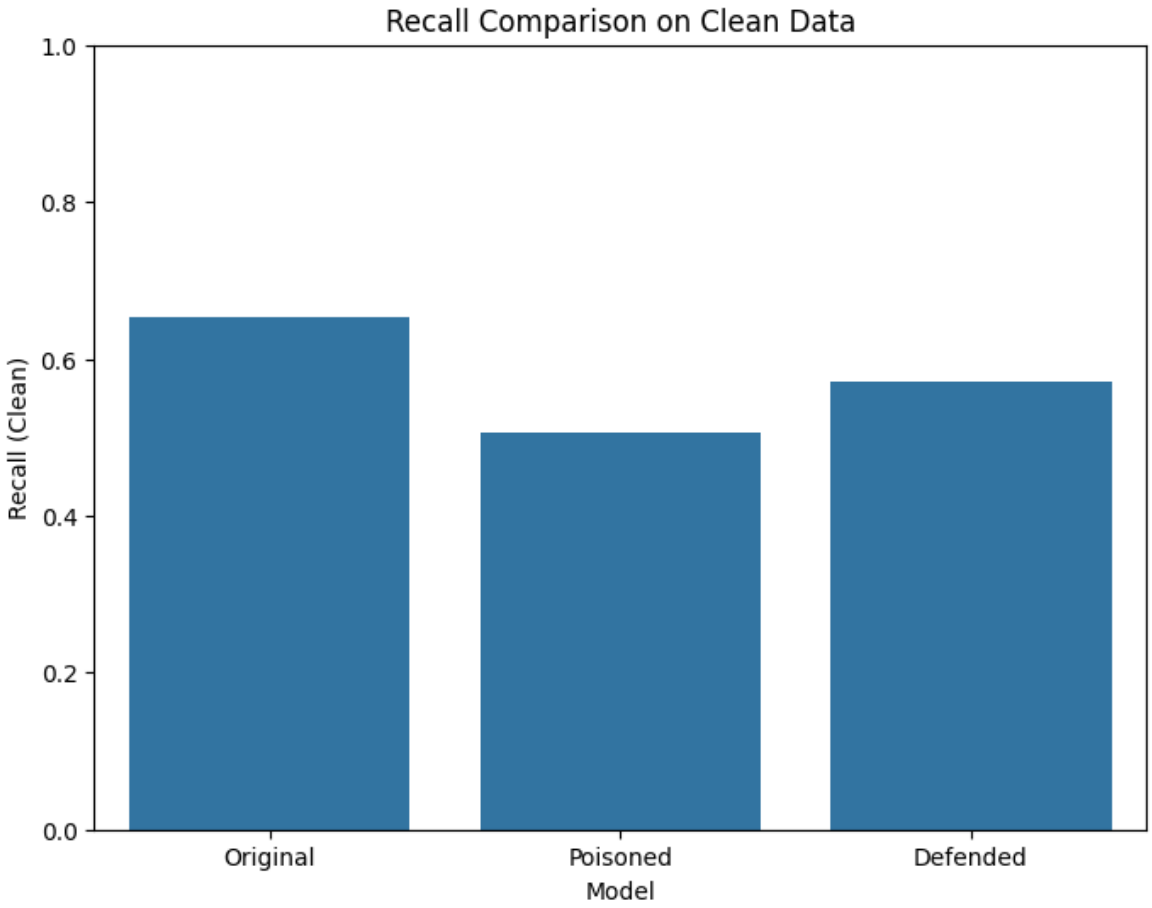
- Mô hình Original đạt được độ chính xác cao nhất (0.6531), cho thấy hiệu suất của mô hình gốc trên dữ liệu sạch là tốt nhất trong ba mô hình.
- Mô hình Poisoned có độ chính xác thấp nhất (0.5059), điều này chỉ ra rằng việc làm nhiễu dữ liệu đã làm giảm đáng kể khả năng dự đoán chính xác của mô hình.
- Mô hình Defended có độ chính xác ở mức trung bình (0.5715), cho thấy các phương pháp phòng thủ đã cải thiện phần nào hiệu suất so với mô hình bị làm nhiễu, nhưng chưa đạt được mức của mô hình gốc.



Hình 36: Biểu đồ Precision

❖ **Nhận xét:**

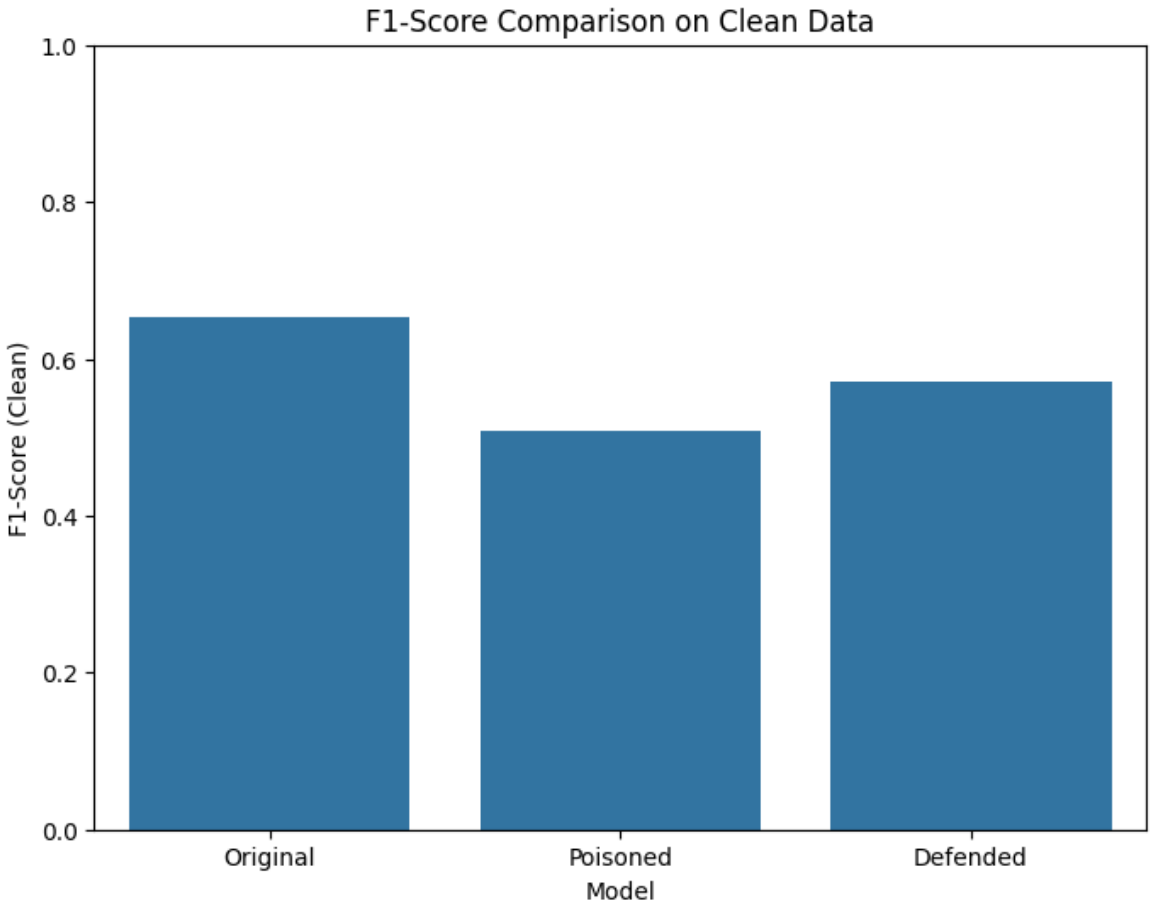
- Mô hình Original có Precision cao nhất (0.660563), chứng minh rằng mô hình gốc đưa ra nhiều dự đoán đúng nhất trên dữ liệu sạch.
- Mô hình Poisoned giảm Precision xuống còn 0.538723, điều này cho thấy việc làm nhiễu dữ liệu khiến mô hình đưa ra nhiều dự đoán sai hơn.
- Mô hình Defended đạt Precision là 0.581554, thể hiện sự cải thiện so với mô hình bị làm nhiễu nhưng vẫn thấp hơn mô hình gốc.



Hình 37: Biểu đồ Recall

❖ **Nhận xét:**

- Original đạt Recall cao nhất (0.6531), cho thấy mô hình gốc nhận diện tốt nhất các nhãn thực sự có trong dữ liệu.
- Poisoned có Recall thấp nhất (0.5059), phản ánh việc làm nhiễu đã làm giảm khả năng nhận diện của mô hình.
- Defended có Recall là 0.5715, cải thiện hơn so với mô hình bị làm nhiễu, nhưng vẫn kém mô hình gốc.



Hình 38: Biểu đồ F1-Score

Nhận xét:

- Original đạt F1-Score cao nhất (0.653159), nhấn mạnh rằng cả Precision và Recall của mô hình gốc đều tốt.
- Poisoned có F1-Score thấp nhất (0.508614), do cả Precision và Recall đều giảm khi mô hình bị làm nhiễu.
- Defended có F1-Score là 0.571553, chứng minh rằng các phương pháp phòng thủ đã giúp cân bằng giữa Precision và Recall, nhưng chưa đạt hiệu suất của mô hình gốc.

3.4. Focal Loss & Hinge Loss

```python
def focal_loss(gamma=4., alpha=0.1, margin=2.):
    def focal_loss_fixed(y_true, y_pred):
        epsilon = K.epsilon()
        y_pred = K.clip(y_pred, epsilon, 1. - epsilon)
        cross_entropy = -y_true * K.log(y_pred)
        focal = alpha * K.pow(1 - y_pred, gamma) * cross_entropy

        # Add Hinge Loss to further mitigate flipped labels
        hinge_loss = K.maximum(0., margin - y_true * y_pred)
        return K.sum(focal + hinge_loss, axis=1)
    return focal_loss_fixed
```

Hình 39: Phương thức phòng thủ Focal Loss & Hinge Loss

```
original_model = ResNet32((32, 32, 3), num_classes).model
original_metrics = train_and_evaluate(original_model, datagen, x_train, y_train, x_test, y_test, "Original Model")
original_history = original_metrics["history"]
original_acc = original_metrics["accuracy"]
original_precision = original_metrics["precision"]
original_recall = original_metrics["recall"]
original_f1_score = original_metrics["f1_score"]
original_time = original_metrics["time"]
original_epochs = original_metrics["epochs"]
```

```
Epoch 1/10
   1/1563 ━━━━━━━━━━━━━━━━━━━━ 5:56:46 14s/step - accuracy: 0.0625 - loss: 3.6716
C:\Users\thevi\AppData\Roaming\Python\Python312\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWar
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 107s 60ms/step - accuracy: 0.2223 - loss: 2.2086 - val_accuracy: 0.4008 - val_loss: 1.6600
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 92s 59ms/step - accuracy: 0.4298 - loss: 1.5795 - val_accuracy: 0.3912 - val_loss: 1.8722
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 91s 58ms/step - accuracy: 0.5174 - loss: 1.3692 - val_accuracy: 0.5455 - val_loss: 1.3267
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 96s 62ms/step - accuracy: 0.5711 - loss: 1.2355 - val_accuracy: 0.5320 - val_loss: 1.4525
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 93s 59ms/step - accuracy: 0.6069 - loss: 1.1345 - val_accuracy: 0.4990 - val_loss: 1.5533
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 94s 60ms/step - accuracy: 0.6345 - loss: 1.0545 - val_accuracy: 0.6658 - val_loss: 0.9872
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 94s 60ms/step - accuracy: 0.6632 - loss: 0.9902 - val_accuracy: 0.6927 - val_loss: 0.8667
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 97s 62ms/step - accuracy: 0.6793 - loss: 0.9477 - val_accuracy: 0.7055 - val_loss: 0.8412
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 99s 63ms/step - accuracy: 0.6966 - loss: 0.9016 - val_accuracy: 0.6644 - val_loss: 1.0166
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 93s 59ms/step - accuracy: 0.7028 - loss: 0.8863 - val_accuracy: 0.7252 - val_loss: 0.7759
313/313 ━━━━━━━━━━━━━━━━━━━━ 5s 15ms/step
Original Model Test Accuracy: 72.51999974250793%
```

Hình 40: Kết quả chạy mô hình ban đầu khi chưa bị tấn công

Sau khi thực hiện chạy mô hình ban đầu, kết quả cho thấy **Accuracy: 72.5%**, kết quả cho ra khá cao do chưa thực hiện bất kỳ cuộc tấn công nào.

```
attacked_model = ResNet32((32, 32, 3), num_classes).model
attacked_metrics = train_and_evaluate(attacked_model, datagen, x_train, y_train_flipped, x_test, y_test, "Attacked Model")
attacked_history = attacked_metrics["history"]
attacked_acc = attacked_metrics["accuracy"]
attacked_precision = attacked_metrics["precision"]
attacked_recall = attacked_metrics["recall"]
attacked_f1_score = attacked_metrics["f1_score"]
attacked_time = attacked_metrics["time"]
attacked_epochs = attacked_metrics["epochs"]
```

```
Epoch 1/10
   2/1563 ━━━━━━━━━━━━━━━━━━━━ 1:24 54ms/step - accuracy: 0.0703 - loss: 3.5470
C:\Users\thevi\AppData\Roaming\Python\Python312\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserW
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 114s 61ms/step - accuracy: 0.1218 - loss: 2.4692 - val_accuracy: 0.3070 - val_loss: 2.0508
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 95s 61ms/step - accuracy: 0.1765 - loss: 2.2400 - val_accuracy: 0.3290 - val_loss: 1.9918
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 96s 61ms/step - accuracy: 0.2000 - loss: 2.2166 - val_accuracy: 0.3747 - val_loss: 1.9426
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 94s 60ms/step - accuracy: 0.2200 - loss: 2.1965 - val_accuracy: 0.4394 - val_loss: 1.8564
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 95s 61ms/step - accuracy: 0.2363 - loss: 2.1762 - val_accuracy: 0.4226 - val_loss: 1.8333
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 95s 61ms/step - accuracy: 0.2510 - loss: 2.1569 - val_accuracy: 0.4620 - val_loss: 1.7323
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 95s 61ms/step - accuracy: 0.2675 - loss: 2.1420 - val_accuracy: 0.5051 - val_loss: 1.6797
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 95s 60ms/step - accuracy: 0.2795 - loss: 2.1259 - val_accuracy: 0.5398 - val_loss: 1.6302
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 99s 63ms/step - accuracy: 0.2847 - loss: 2.1230 - val_accuracy: 0.5381 - val_loss: 1.6386
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 98s 63ms/step - accuracy: 0.2968 - loss: 2.1066 - val_accuracy: 0.5709 - val_loss: 1.5631
313/313 ━━━━━━━━━━━━━━━━━━━━ 6s 17ms/step
Attacked Model Test Accuracy: 57.0900022983551%
```

Hình 41: Kết quả chạy mô hình sau khi bị tấn công

Sau khi thực hiện chạy mô hình bị tấn công bởi label flipping, kết quả cho thấy **Accuracy: 57.09%**, kết quả cho ra khá thấp so với mô hình ban đầu, do đã bị tấn công khiến cho khả năng phân loại của mô hình giảm đáng kể.

```
defended_model = ResNet32((32, 32, 3), num_classes).model
defended_model.compile(optimizer='adam', loss=focal_loss(gamma=4., alpha=0.1, margin=2.), metrics=['accuracy'])
defended_metrics = train_and_evaluate(defended_model, datagen, x_train, y_train_flipped, x_test, y_test, "Defended Model")
defended_history = defended_metrics["history"]
defended_acc = defended_metrics["accuracy"]
defended_precision = defended_metrics["precision"]
defended_recall = defended_metrics["recall"]
defended_f1_score = defended_metrics["f1_score"]
defended_time = defended_metrics["time"]
defended_epochs = defended_metrics["epochs"]
```

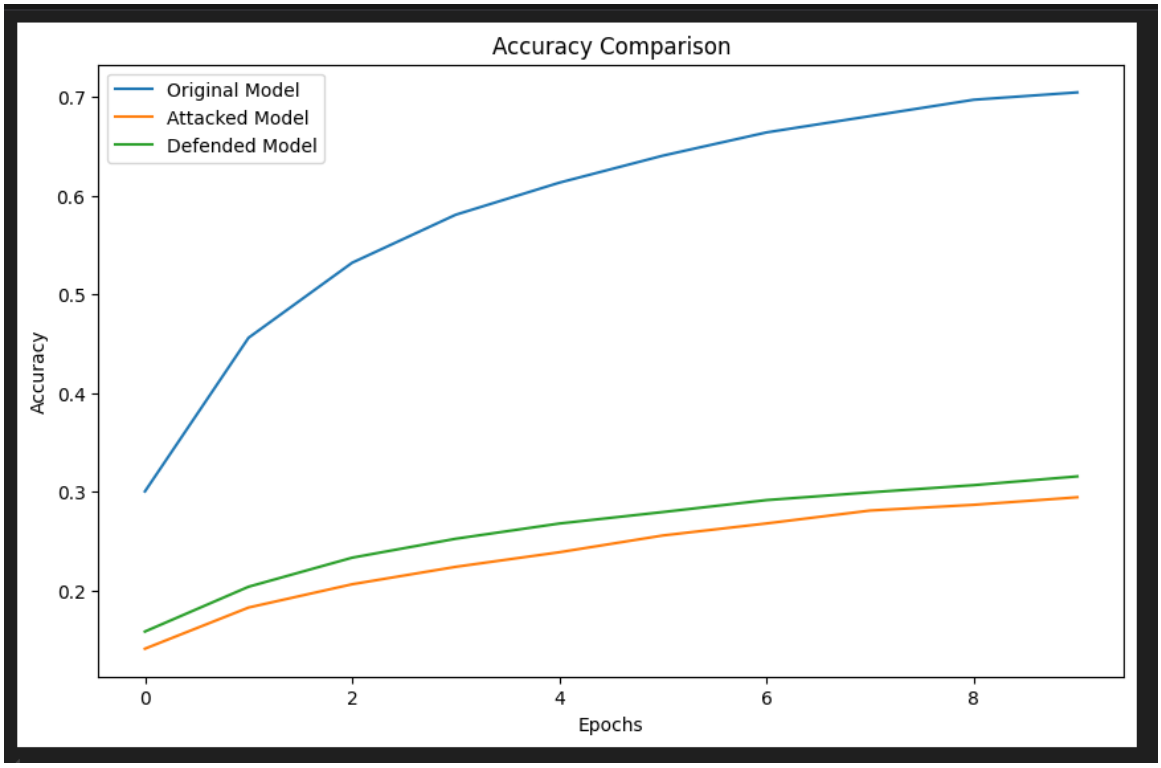
```
Epoch 1/10
  1/1563 ━━━━━━━━━━━━━━━━━━━━ 6:32:55 15s/step - accuracy: 0.1250 - loss: 20.2598
C:\Users\thevi\AppData\Roaming\Python\Python312\site-packages\keras\src\trainers\data_adapters\py_dataset_adapter.py:121: UserWa
  self._warn_if_super_not_called()
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 142s 81ms/step - accuracy: 0.1375 - loss: 20.0717 - val_accuracy: 0.2796 - val_loss: 19.9420
Epoch 2/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.1966 - loss: 20.0171 - val_accuracy: 0.3782 - val_loss: 19.8626
Epoch 3/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.2284 - loss: 20.0028 - val_accuracy: 0.4593 - val_loss: 19.7919
Epoch 4/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.2464 - loss: 19.9900 - val_accuracy: 0.3590 - val_loss: 19.8741
Epoch 5/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.2659 - loss: 19.9789 - val_accuracy: 0.3812 - val_loss: 19.8395
Epoch 6/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.2796 - loss: 19.9688 - val_accuracy: 0.5300 - val_loss: 19.6983
Epoch 7/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.2904 - loss: 19.9614 - val_accuracy: 0.4860 - val_loss: 19.7283
Epoch 8/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.2943 - loss: 19.9585 - val_accuracy: 0.5203 - val_loss: 19.6801
Epoch 9/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 129s 82ms/step - accuracy: 0.3062 - loss: 19.9492 - val_accuracy: 0.5160 - val_loss: 19.6986
Epoch 10/10
1563/1563 ━━━━━━━━━━━━━━━━━━━━ 122s 78ms/step - accuracy: 0.3179 - loss: 19.9401 - val_accuracy: 0.5863 - val_loss: 19.6357
313/313 ━━━━━━━━━━━━━━━━━━━━ 7s 22ms/step
Defended Model Test Accuracy: 58.63000154495239%
```

Hình 42: Kết quả chạy mô hình sau khi áp dụng phòng thủ Focal Loss & Hinge Loss

Sau khi áp dụng phương pháp phòng thủ vào tập dữ liệu bị tấn công, kết quả **Accuracy: 58.6%,** mô hình phân loại đã có sự cải thiện nhưng không đáng kể so với kết quả khi phân loại trên mô hình bị tấn công.

	Model	Accuracy	Precision	Recall	F1-Score	Epoch Time (s)
0	Original	72.520000	0.727303	0.7252	0.719865	10
1	Attacked	57.090002	0.567307	0.5709	0.547631	10
2	Defended	58.630002	0.591355	0.5863	0.562637	10

Hình 43: Kết quả so sánh dạng bảng của 3 mô hình ban đầu, mô hình bị tấn công, mô hình phòng thủ

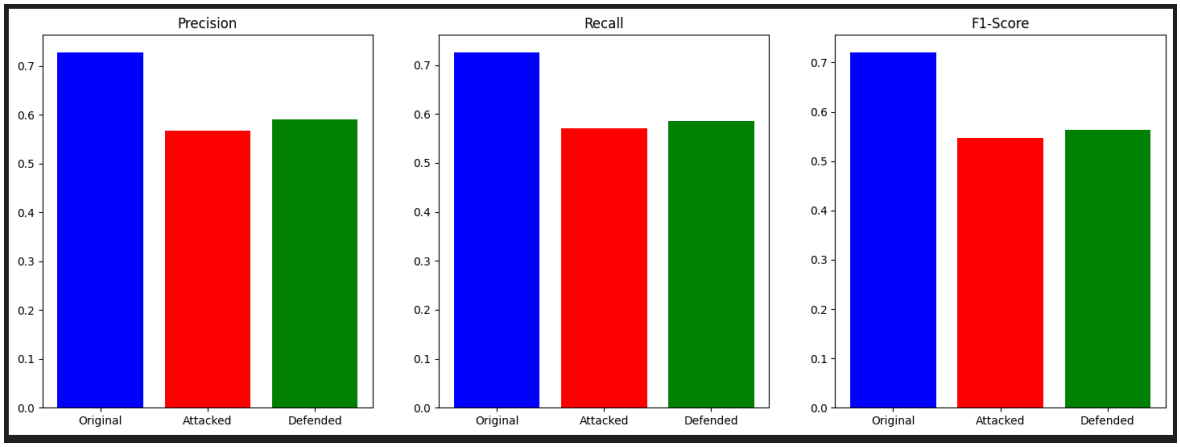


Hình 44: Trực quan hóa biểu đồ tương quan giữa Accuracy và Epochs của 3 mô hình

Nhận thấy mô hình ban đầu có accuracy khá cao ngay từ ban đầu và từ sau epochs đầu tiên độ chính xác của mô hình tăng mạnh, sau đó tăng từ từ đến hết epochs.

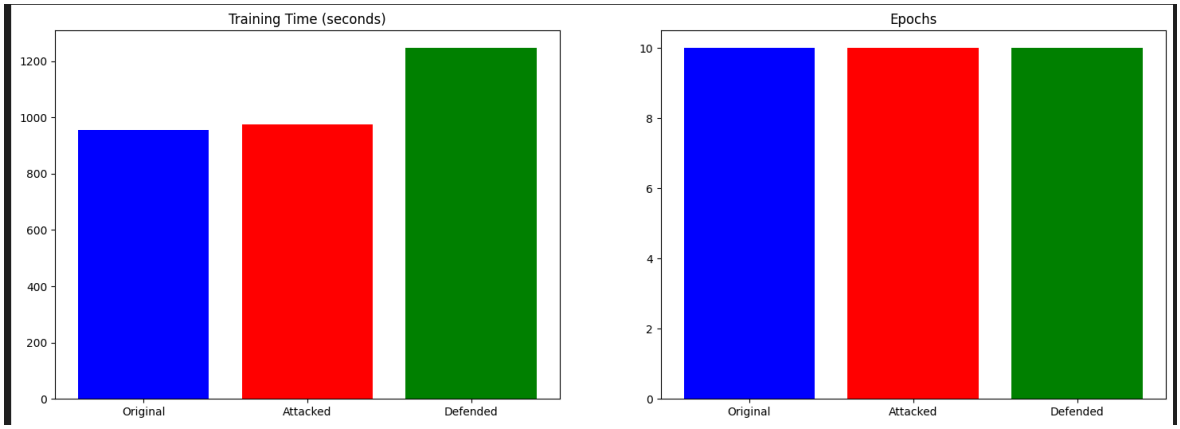
Mô hình bị tấn công có accuracy rất thấp ngay từ ban đầu nhưng vẫn tăng từ từ đến hết epochs.

Mô hình phòng thủ có accuracy ban đầu cải thiện hơn so với mô hình bị tấn công và tiếp tục tăng từ từ đến hết epochs.



Hình 45: Trực quan hóa dạng biểu đồ các thông số Precision, Recall, F1-Score của 3 mô hình

Nhận thấy kết quả của cả ba chỉ số rất đồng đều, mô hình ban đầu thể hiện độ chính xác vượt trội so với 2 mô hình còn lại, mô hình phòng thủ mặc dù các thông số cao hơn mô hình tấn công nhưng sự chênh lệch này là không đáng kể.



Hình 46: Trực quan hóa dạng biểu đồ các thông số Training time, epochs của 3 mô hình

Kết quả với cùng 1 thông số epochs, cho thấy mô hình phòng thủ cần nhiều thời gian hơn để thực hiện do áp dụng thêm kĩ thuật phòng thủ so với 2 mô hình còn lại, mô hình tấn công có thời gian chạy lâu hơn mô hình ban đầu nhưng sự vượt trội này rất thấp.

❖ **Kết luận:**

- Các mô hình đã đạt được các thông số đúng theo lý thuyết, mô hình ban đầu sẽ có thông số tốt nhất, tiếp theo là mô hình phòng thủ, cuối cùng là mô hình tấn công.

- Nhưng kết quả của mô hình phòng thủ không đạt được thông số như kỳ vọng, mức độ cải thiện còn rất thấp so với khi bị tấn công
- Cần cải thiện lại công thức, tối ưu hóa lại tham số hoặc sử dụng phương pháp phòng thủ khác để đạt kết quả khả quan hơn.

4. THAM KHẢO

[1] *"Focal Loss for Dense Object Detection. (n.d)"*. Available at https://ar5iv.org/html/1708.02002 [Accessed 05/12/2024]

[2] *"Defending against the Label-flipping Attack in Federated Learning. (n.d)"*. Available at https://ar5iv.org/html/2207.01982 - [Accessed 04/12/2024]

[3] *"Review for NeurIPS paper: On the Error Resistance of Hinge-Loss Minimization. (n.d.)"*. Available at https://proceedings.neurips.cc/paper_files/paper/2020/file/2c5201a7391fedbc40c3cc6aa057a029-Review.html - [Accessed 04/12/2024]