Evaluating Model Robustness to Adversarial Attacks

Chijioke Ugwuanyi

24th April 2025

1 Problem Statement

Deep learning models, particularly convolutional neural networks (CNNs), excel in image classification tasks but are vulnerable to adversarial attacks, which are small, often imperceptible perturbations to input data that lead to incorrect predictions. This analysis investigates the vulnerability of deep learning models to adversarial attacks and explores the effectiveness of adversarial training as a defense mechanism.

Specifically, the study focuses on image classification using the CIFAR-10 dataset and evaluates the robustness of a ResNet50 model, both naturally trained and adversarially trained, against Fast Gradient Sign Method (FGSM) attacks.

Additionally, the robustness of other pre-trained models, namely VGG16, MobileNetV2, and EfficientNetB0, to FGSM attacks is also examined and evaluated to identify the most robust. The goal is to quantify the impact of adversarial perturbations on model accuracy and to assess the potential of adversarial training in mitigating these effects.

2 Data Source

The CIFAR-10 dataset is used for this study. This dataset comprises 60,000 32×32 color images across 10 different classes (e.g., airplane, car, dog, truck). For computational efficiency, a subset of the data is used: 5,000 training samples and 1,000 validation samples derived from the original test set. The dataset is accessed via TensorFlow's `tensorflow.keras.datasets.cifar10` module. The pixel values are normalized to [0, 1] by dividing by 255. Labels are one-hot encoded. The images are used for training and evaluating the image classification models.

3 Exploratory Data Analysis

Given CIFAR-10's structured nature, exploratory data analysis (EDA) focused on understanding the dataset's properties:

- **Class Distribution:** The subset maintains a balanced distribution, with approximately 500 training and 100 validation images per class.

- **Image Characteristics:** Images are low-resolution (32×32), which poses challenges for models pre-trained on higher-resolution datasets (e.g., ImageNet). Resizing to 224×224 introduces interpolation artifacts but enables transfer learning.
- No significant data quality issues were identified, though resizing impacts fine-grained details, potentially affecting model performance.

4 Feature Engineering

Feature engineering was minimal due to the use of pre-trained CNNs, which inherently extract features. However, the following were applied:

- **Image Resizing:** Images were resized from 32×32 to 224×224 using TensorFlow's resize function to match model input requirements.
- **Normalization:** Pixel values were scaled to [0, 1], aligning with pre-trained model expectations.
- **Data Augmentation:** No augmentation was applied during training to focus on adversarial robustness, but ImageDataGenerator was used for batching.
- **Adversarial Examples:** For the adversarially trained ResNet50, FGSM-generated adversarial examples (epsilon=0.05) were mixed with clean data (50/50 ratio) during training to improve robustness.

5 Modeling Process

The analysis involves training and evaluating several convolutional neural network (CNN) architectures:

- **ResNet50 (Naturally Trained):** A pre-trained ResNet50 model (weights from ImageNet) is used as a base. The top classification layer is excluded, and a new classification head tailored for the 10 CIFAR-10 classes is added. Some of the later layers of the base model are unfrozen to allow for fine-tuning on the CIFAR-10 dataset. The model is trained using the Adam optimizer with a learning rate of 1e-5 and categorical cross-entropy loss for 20 epochs, with early stopping and learning rate reduction callbacks.
- **Adversarial Attack Generation (FGSM):** The Fast Gradient Sign Method (FGSM) is implemented to create adversarial examples. This technique computes the gradient of the loss function concerning the input image and adds a small perturbation (controlled by an epsilon parameter, set to 0.05 in the primary experiments) in the direction of the sign of this gradient. The goal is to create images that are perceptually similar to the original but can fool the trained model into making incorrect predictions.

- **ResNet50 (Adversarially Trained):** A new ResNet50 model with a similar architecture to the naturally trained one is subjected to adversarial training. In this process, during each training epoch, a batch of clean images is taken, and corresponding adversarial examples are generated using the FGSM attack. The model is then trained on a mixed dataset containing both the original clean examples and the generated adversarial examples. This process aims to make the model more robust to adversarial perturbations.

- **Evaluation on Clean and Adversarial Examples:** Both the naturally trained and adversarially trained ResNet50 models are evaluated on a clean validation set and on adversarial examples generated from the validation set using the FGSM attack with an epsilon of 0.05. The fooling rate is calculated to quantify the decrease in accuracy due to the adversarial attack.

- **Comparison with Other Pre-trained Models (VGG16, MobileNetV2, EfficientNetB0):** Three other pre-trained models (VGG16, MobileNetV2, EfficientNetB0) are used as fixed feature extractors. Their base convolutional layers are frozen, and a new classification head is added and trained on the CIFAR-10 subset. These models are also evaluated on both clean and FGSM adversarial examples (epsilon=0.05) to compare their inherent robustness.

- **Analysis of Accuracy vs. Epsilon:** The performance of the adversarially trained ResNet50 model is assessed under FGSM attacks with varying epsilon values (0.01, 0.05, 0.1, 0.2, 0.3) to understand how the strength of the perturbation affects its accuracy.

6 Modeling Results

6.1 Baseline Performance

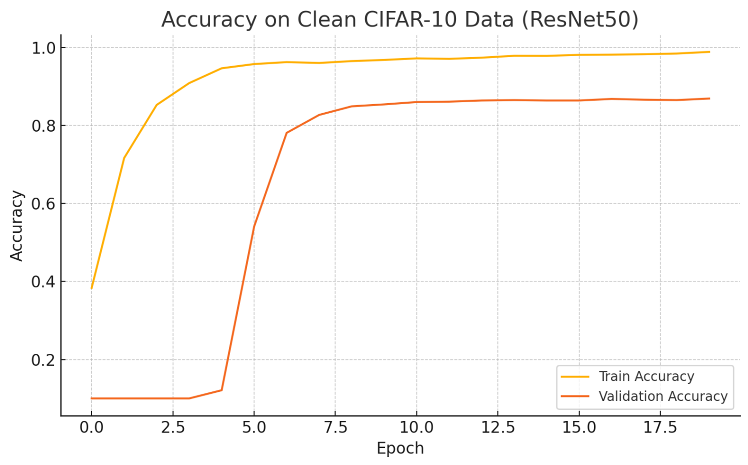

Figure 1: Accuracy on Clean CIFAR-10 Data (ResNet50)

A ResNet50 basemodel was trained on a clean subset of CIFAR-10 data from Kaggle. It achieved a baseline performance of 86.9% on the clean data.

6.2 Epsilon=0.05

Adversarial Training Performance

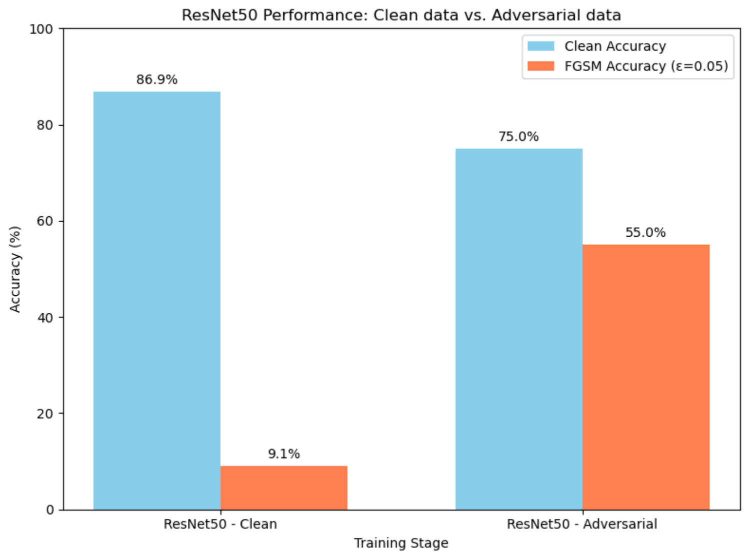

Figure 2: ResNet50 Performance: Clean data vs. Adversarial data

- With a perturbation magnitude of 0.05, and using the Fast Gradient Sign Method (FGSM), adversarial data was generated, and the base model with an accuracy of 86.9% was re-evaluated on the adversarial data. The accuracy of the model went down to 9.1%, showing how vulnerable the model is to adversarial input.
- On the other hand, another ResNet50 model was created using the same architecture as the first model. It was trained using a 50-50 mixture of clean and adversarial data. This model was then evaluated on both the clean and adversarial data.
  - It achieved a clean data accuracy of 75% because of the noise in the adversarial data used for its training.
  - It achieved an FGSM accuracy of 55%, which is relatively poor, however, 55% is a giant leap from 9.1 for the model trained solely on clean data.

  This shows how adversarial training could improve model accuracy.

6.3 Models Comparison

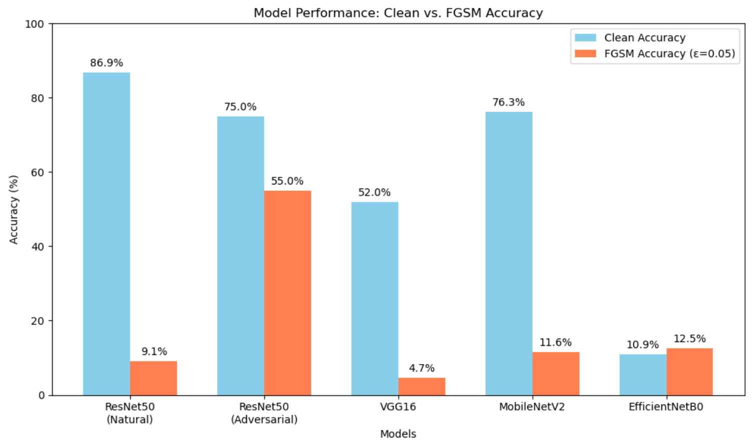

Figure 3: Model Performance: Clean vs. FGSM Accuracy

3 new models: VGG16, MobileNetV2, and EfficientNetB0, were investigated for their robustness to adversarial attack without adversarial training. They were trained exclusively on clean data and evaluated on adversarial data. The following insights were gained:

- ResNet50 (Natural) has the highest clean accuracy (86.90%) but performs poorly under an FGSM attack (9.06%), showing its vulnerability to adversarial examples.
- ResNet50 (Adversarial), after mixed training, achieves a balanced performance with ∼75% clean and ∼55% FGSM accuracy, making it the most robust overall.
- MobileNetV2 performs well on clean data (76.30%) but struggles with FGSM (11.56%), similar to ResNet50 (Natural).
- VGG16 shows moderate clean accuracy (52.00%) but very poor FGSM performance (4.69%).
- EfficientNetB0 performs poorly on both clean (10.90%) and FGSM (12.50%), indicating it's not suitable for this task without significant adjustments.
- Due to the ineffectiveness of EfficientNetB0, MobileNetV2 seems to be the most robust without adversely training.

6.4 Epsilon Sensitivity (ResNet50 Adversarial)

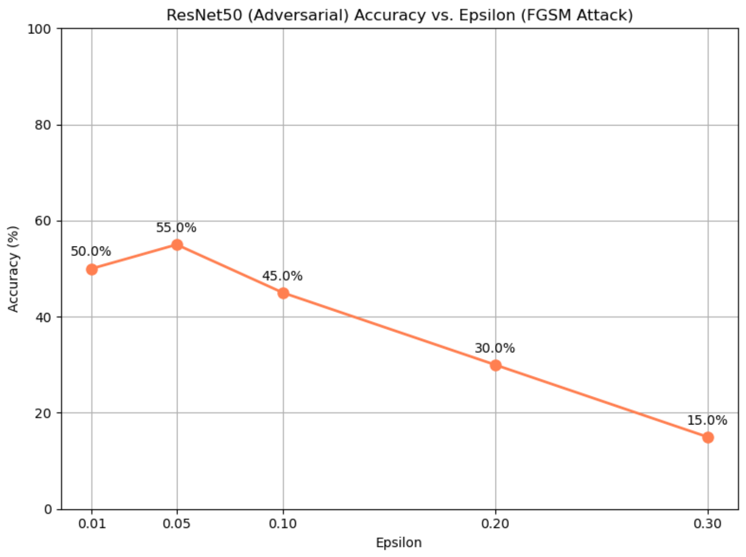

Figure 4: ResNet50 (Adversarial) Accuracy vs. Epsilon (FGSM Attack)

As the epsilon value increased, the accuracy of the adversarially trained model generally decreased, as expected. However, it maintained a relatively high accuracy for smaller epsilon values, highlighting the benefits of adversarial training for defending against weaker attacks.

7 Main Insights

The analysis reveals a critical insight into the security of deep learning models: models trained solely on clean data are highly susceptible to adversarial attacks. The significant drop in accuracy of the naturally trained ResNet50 model under FGSM perturbations shows this vulnerability.

However, the study also demonstrates that adversarial training is a promising technique for enhancing the robustness of models against these attacks. The adversarially trained ResNet50 model exhibited a remarkable ability to maintain high accuracy even when presented with adversarial examples, although this improvement in robustness came at a cost of reduced performance on clean, unperturbed data.

Furthermore, the comparison with other pre-trained architectures suggests that different models possess varying inherent levels of robustness to adversarial attacks, even without specific adversarial training. These findings indicate the importance of considering adversarial robustness as a key evaluation metric for deep learning models, especially in security-sensitive applications.

The trade-off between clean accuracy and adversarial robustness is an important consideration in deploying deep learning models in real-world scenarios where adversarial inputs might be encountered.

A Appendix

A.1 Data Source

```
# Load and preprocess CIFAR-10 data (subset)
(x_train, y_train), (x_test, y_test) = cifar10.load_data()
x_train, x_test = x_train.astype('float32') / 255.0, x_test.astype('
    float32') / 255.0
y_train, y_test = tf.keras.utils.to_categorical(y_train, 10), tf.keras.
    utils.to_categorical(y_test, 10)
```

A.2 Feature Engineering

```
# Use a subset to reduce compute
x_train, y_train = x_train[:5000], y_train[:5000]
x_val, y_val = x_test[:1000], y_test[:1000]

# Resize images for ResNet50 (224x224)
x_train = resize(x_train, [224, 224])
x_val = resize(x_val, [224, 224])

# Data generators
train_datagen = ImageDataGenerator()
val_datagen = ImageDataGenerator()
train_generator = train_datagen.flow(x_train, y_train, batch_size=
    BATCH_SIZE)
val_generator = val_datagen.flow(x_val, y_val, batch_size=BATCH_SIZE)
```

A.3 Modelling Process

```
# Build ResNet50 model
base_model = ResNet50(weights='imagenet', include_top=False, input_shape
    =(224, 224, 3))
for layer in base_model.layers[-30:]:
    layer.trainable = True # Unfreeze some base layers to reduce compute

x = base_model.output
x = GlobalAveragePooling2D()(x)
x = Dense(1024, activation='relu')(x)
x = Dropout(0.3)(x)
predictions = Dense(10, activation='softmax')(x)
model = Model(inputs=base_model.input, outputs=predictions)

callbacks = [
    EarlyStopping(patience=5, restore_best_weights=True),
    ReduceLROnPlateau(factor=0.1, patience=3)
]

# Compile and train
model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
              loss='categorical_crossentropy', metrics=['accuracy'])
```

A.4 Adversarial Attack and Evaluation

```python
# FGSM attack implementation
def fgsm_attack(model, images, labels, epsilon):
    images = tf.cast(images, tf.float32)
    labels = tf.cast(labels, tf.float32)
    with tf.GradientTape() as tape:
        tape.watch(images)
        predictions = model(images)
        loss = tf.keras.losses.categorical_crossentropy(labels,
    predictions)
    gradient = tape.gradient(loss, images)
    signed_grad = tf.sign(gradient)
    adv_images = images + epsilon * signed_grad
    adv_images = tf.clip_by_value(adv_images, 0, 1)
    return adv_images

# Evaluate on adversarial examples
def evaluate_adversarial(model, generator, epsilon):
    total_loss, total_acc = 0, 0
    num_batches = 0
    generator.reset()
    for images, labels in generator:
        adv_images = fgsm_attack(model, images, labels, epsilon)
        loss, acc = model.evaluate(adv_images, labels, verbose=1)
        total_loss += loss
        total_acc += acc
        num_batches += 1
        if num_batches >= 10: # Limit to 10 batches
            break
    avg_loss = total_loss / num_batches
    avg_acc = total_acc / num_batches
    return avg_loss, avg_acc

# Evaluate FGSM with epsilon=0.05
epsilon = 0.05
fgsm_loss, fgsm_acc = evaluate_adversarial(model, val_generator, epsilon)
print(f"FGSM Attack (epsilon={epsilon}) - Loss: {fgsm_loss:.4f}, Accuracy:
    {fgsm_acc:.4f}")

# Calculate fooling rate
clean_loss, clean_acc = model.evaluate(val_generator, verbose=1)
fooling_rate = (clean_acc - fgsm_acc) / clean_acc * 100
print(f"Fooling Rate: {fooling_rate:.2f}%")
```

A.5 Models Comparison

```python
from tensorflow.keras.applications import VGG16, MobileNetV2,
    EfficientNetB0

# Function to build and train a model
def build_and_train_model(base_model_class, model_name):
    base_model = base_model_class(weights='imagenet', include_top=False, input_shape=(224, 224, 3))
    for layer in base_model.layers:
        layer.trainable = False
    x = base_model.output
    x = GlobalAveragePooling2D()(x)
    x = Dense(1024, activation='relu')(x)
    predictions = Dense(10, activation='softmax')(x)
    model = Model(inputs=base_model.input, outputs=predictions)

    model.compile(optimizer=tf.keras.optimizers.Adam(learning_rate=1e-4),
                  loss='categorical_crossentropy', metrics=['accuracy'])
    model.fit(train_generator, epochs=EPOCHS, validation_data=val_generator, verbose=1)

    # Evaluate
    val_generator.reset()
    clean_loss, clean_acc = model.evaluate(val_generator, verbose=0)
    fgsm_loss, fgsm_acc = evaluate_adversarial(model, val_generator, epsilon=EPSISON)

    print(f"\n{model_name}:")
    print(f"Clean Data - Loss: {clean_loss:.4f}, Accuracy: {clean_acc:.4f}")
    print(f"FGSM Attack (epsilon={EPSISON}) - Loss: {fgsm_loss:.4f}, Accuracy: {fgsm_acc:.4f}")
    return clean_acc, fgsm_acc

# Train and evaluate VGG16, MobileNetV2, and EfficientNetB0 (as a YOLO-like backbone)
vgg_clean, vgg_fgsm = build_and_train_model(VGG16, "VGG16")
mobilenet_clean, mobilenet_fgsm = build_and_train_model(MobileNetV2, "MobileNetV2")
efficientnet_clean, efficientnet_fgsm = build_and_train_model(EfficientNetB0, "EfficientNetB0 (YOLO-like)")
```

A.6 Epsilon Sensitivity

```
epsilons = [0.01, 0.05, 0.1, 0.2, 0.3]
accuracies = []
for epsilon in epsilons:
    val_generator.reset()
    _, acc = evaluate_adversarial(adv_model, val_generator, epsilon)
    accuracies.append(acc)
    print(f"Epsilon: {epsilon}, Accuracy: {acc:.4f}")
```