MINISTRY OF EDUCATION AND TRAINING
CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY

PROJECT – FUNDAMENTAL TOPICS IN INFORMATION TECHNOLOGY

Topic

ADVERSARIAL ROBUSTNESS OF CIFAR-10 MODEL

Student: Truong Dang Truc Lam
ID: B2111933
Course: K47

Can Tho, 02/2024

MINISTRY OF EDUCATION AND TRAINING
CAN THO UNIVERSITY
COLLEGE OF INFORMATION AND COMMUNICATION TECHNOLOGY
DEPARTMENT OF INFORMATION TECHNOLOGY

PROJECT – FUNDAMENTAL TOPICS IN INFORMATION TECHNOLOGY

Topic

ADVERSARIAL ROBUSTNESS OF CIFAR-10 MODEL

Advisor:
Dr. Pham The Phi

Student:
Truong Dang Truc Lam
ID: B2111933
Course: K47

Can Tho, 02/2014

ACKNOWLEDGEMENTS

I would first like to thank my advisor, Dr. Pham The Phi. Without his assistance and dedicated involvement in every step throughout the process, this paper could have never been accomplished. Moreover, his knowledge and experience have inspired me throughout my academic career and everyday life.

I would also like to thank my supervisor Dr. Lam Nhut Khang for sharing expertise, giving valuable guidance and encouragement extended to me from first year until now.

I take this opportunity to express gratitude to all of The College of Information and Communication Technology members for their help and support.

Finally, I must express my very profound gratefulness to my family for constantly encouraging and supporting me during my years of studies.

Author

Truong Dang Truc Lam

CONTENTS

TABLE OF IMAGES

INTRODUCTION

1. Problems

Thanks to recent breakthroughs in computer vision [1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (NeurIPS), 2012., trained classifiers are being brought into the center of security-critical systems. Vision for autonomous cars, face recognition, and malware detection are some of notable examples. The importance of security aspects of machine learning is increasing as a result of these advancements. Specifically, resistance to adversarially chosen inputs is becoming a crucial design goal. While trained models tend to be very effective in classifying benign inputs, recent research [2] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In Conference on computer vision and pattern recognition (CVPR), 2015. indicates that an adversary is often able to manipulate the input so that the model produces an incorrect output.

This topic has received particular attention in the context of deep neural networks [3] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. Machine Learning, 107(3):481–508, 2018.[4] Jure Sokoli´c, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. In Transactions on Signal Processing, 2017.. Computer vision has to confront a particularly challenge: very little changes to the input image can even fool state-of-the-art neural networks with high confidence [5] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In Computer Vision and Pattern Recognition (CVPR), 2016.. This holds even when the benign example was classified correctly, and human would not be able to notice the change. Apart from the security implications, this phenomenon also demonstrates that our current models are not learning the underlying concepts in a robust manner. All these findings raise a fundamental question: How can we train deep neural networks that are robust to adversarial inputs?

2. Object

Recently, there has been much progress on adversarial attacks against neural networks. Especially CleverHans [6] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, Rujun Long, Patrick McDaniel: Technical Report on the

CleverHans v2.1.0 Adversarial Examples Library, 2018, a Python library for benchmarking the vulnerability of machine learning models to adversarial examples. The library focuses on providing reference implementation of attacks against machine learning models to help with benchmarking models against adversarial examples.

3. Research scope

CIFAR-10 dataset [7] Alex Krizhevsky: Learning Multiple Layers of Features from Tiny Images, 2009..

4. Research method

This research includes training a Convolutional Neural Network (CNN) [8] Dan C. Ciresan, Ueli Meier, Jonathan Masci, et al. Flexible, High Performance Convolutional Neural Networks for Image Classification[J]. PROCEEDINGS OF THE TWENTY-SECOND INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2011. to classify CIFAR images and craft adversarial examples using the Fast Gradient Sign Method (FGSM) [9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations (ICLR), 2015. and Projected Gradient Descent (PGD) [10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu: Towards Deep Learning Models Resistant to Adversarial Attacks, 2019..

MAIN CONTENTS

CHAPTER 1: REQUIREMENT SPECIFICATION

1. Introduction

This document outlines the requirements for a system that evaluates and improves the adversarial robustness of a machine learning model trained on the CIFAR-10 dataset. Adversarial robustness refers to a model's ability to resist attacks from adversarial examples (slightly modified inputs that cause incorrect predictions).

2. System Description

The system will assess and enhance the adversarial robustness of a pre-trained image classification model on the CIFAR-10 dataset.

3. Requirements

3.1 Functional Requirements

Input:

- The system shall accept a pre-trained image classification model as input. The model shall be compatible with a common deep learning framework (e.g.,TensorFlow, PyTorch).
- The system shall accept the CIFAR-10 dataset as input.

Evaluation:

- The system shall generate adversarial examples for the CIFAR-10 dataset using configurable attack algorithms (e.g., FGSM, PGD).
- The system shall evaluate the success rate of the adversarial attacks in fooling the target model.
- The system shall provide metrics to quantify the adversarial robustness of the model, such as the percentage of adversarial examples generated.

Improvement:

- The system shall allow integration of adversarial training techniques to improve the model's robustness, such as adding adversarial examples to the training data.
- The system shall be configurable to adjust training hyperparameters (e.g., epsilon, epochs) during adversarial training.

3.2 Non-Functional Requirements

Performance: The system shall evaluate the model's robustness and perform adversarial training efficiently.

Configurability: The system shall be configurable to allow for different attack algorithms, training techniques, and hyperparameter settings.

Usability: The system shall have a user-friendly interface (command line or API) for ease of use.

4. Acceptance Criteria

- The system shall successfully evaluate the pre-trained model's robustness against adversarial attacks using configurable metrics.
- The system shall enable adversarial training to improve the model's robustness, demonstrably reducing the success rate of adversarial attacks.
- The system shall be modular and allow for easy integration of new attack algorithms and training techniques.

CHAPTER 2: THEORETICAL BASIS

1. Python

Python is a great choice for building machine learning models due to its ease of use, extensive framework library, flexibility and more. Python is also compatible with a wide range of operating systems, including Windows, Linux, Unix, and macOS. Most importantly, a large community of developers adore Python for being a simple language to read. In short, Python’s online community makes it easy to find answers and resources when building or troubleshooting machine learning models.

Figure 1. Python

2. TensorFlow

TensorFlow is a powerful and popular open-source library for machine learning and artificial intelligence. This versatile toolkit provides the building blocks, code and data to craft these intelligent minds, specializing in a powerful technique called deep learning. Tensorflow is used by a wide range of people, from researchers and developers to businesses and hobbyists, to build and train intelligent systems. TensorFlow offers a beginner-friendly interface, pre-built models and lets you deploy your creations in real-world settings.

Figure 2: Tensorflow

3. Cleverhans

CleverHans is a software library that provides standardized reference implementations of adversarial example construction techniques and adversarial training. The library may be used to develop more robust machine learning models and to provide standardized benchmarks of models' performance in the adversarial setting. Benchmarks constructed without a standardized implementation of adversarial example construction are not comparable to each other, because a good result may indicate a robust model or it may merely indicate a weak implementation of the adversarial example construction procedure.

Figure 3. CleverHans library

4. CIFAR-10 dataset

CIFAR-10 is a popular dataset used to train and test machine learning algorithms, particularly in the field of image recognition. It consists of 60,000 32x32 pixel color images belonging to 10 different classes: airplanes, automobiles, birds, cats, deer, dogs, frogs, horses, ships, and trucks. CIFAR-10 is often used as a benchmark for testing the performance of image recognition algorithms. It is a good starting point for researchers and developers who are new to the field of deep learning.

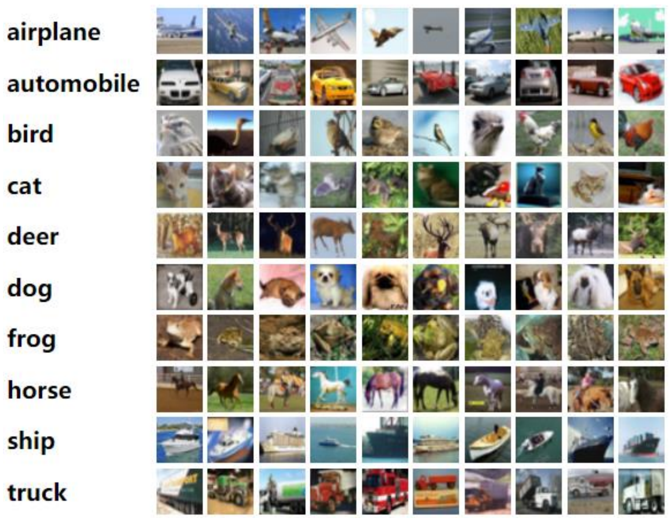


Figure 4: CIFAR-10

5. Convolutional Neural Network (CNN) model

In the field of artificial intelligence, Convolutional Neural Networks (CNNs) represent a specific type of artificial neural network architecture designed to excel in visual recognition and analysis. Their design draws inspiration from the visual processing pathway found in the human brain, leading to remarkable capabilities in extracting meaningful features from visual data like images and videos.

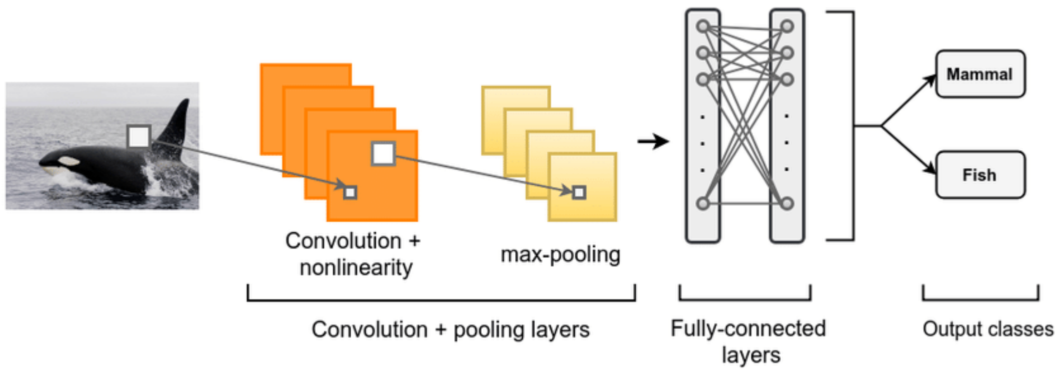


Figure 5: CNN model

6. Adversarial examples

Within machine learning, adversarial examples are meticulously crafted inputs designed to deceive models, causing them to make incorrect predictions with high confidence. Imagine a photo of a cat, subtly altered with almost imperceptible noise, yet tricking the model into classifying it as a dog. These seemingly minor changes exploit the model's internal decision-making boundaries, highlighting potential vulnerabilities. While seemingly benign, adversarial examples raise concerns about the security and robustness of models employed in critical domains like autonomous vehicles and medical diagnosis.

6.1. Fast Gradient Sign Method (FGSM)

The Fast Gradient Sign Method (FGSM) is a white-box adversarial attack in machine learning, aiming to craft tiny input modifications that cause models to make incorrect predictions with high confidence. Operating under the assumption of full knowledge about the target model, FGSM leverages gradients, directional guides in the loss landscape, to manipulate the input image. It calculates the gradient with respect to the input, then simplifies it by using only the sign (+/- 1), leading to an efficient sign-flipping step. This modified gradient is scaled and added to the original image, creating an adversarial example designed to fool the model. While efficient and effective for targeted attacks, FGSM's adversarial examples may lack transferability and struggle with crafting highly complex perturbations. Nevertheless, its simplicity and power make it a cornerstone technique for understanding and mitigating adversarial vulnerabilities in machine learning models.

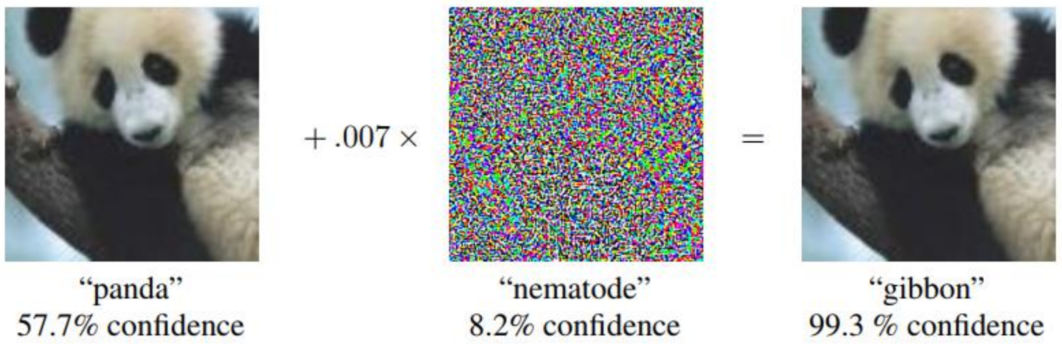


Figure 6: FGSM example

6.2. Projected Gradient Descent (PGD)

Projected Gradient Descent (PGD) is a white-box attack which means the attacker has access to the model gradients or the attacker has a copy of the weights of your model. This threat model gives the attacker much more power than black box attacks as they can specifically craft their attack to fool your model without having to rely on transfer attacks that often result in human-visible perturbations. PGD can be considered the most "complete" white-box adversary as it lifts any constraints on the amount of time and effort the attacker can put into finding the best attack.

The key to understanding the PGD attack is to frame finding an adversarial example as a contrained optimisation problem. PGD attempts to find the perturbation that maximises the loss of a model on a particular input while keeping the size of the perturbation smaller than a specified amount referred to as epsilon. This constraint is typically expressed as the L^2 or $L\infty$ norm of the perturbation. It will be applied so the content of the adversarial example is the same as the unperturbed sample, or even such that the adversarial example is imperceptibly different to humans.

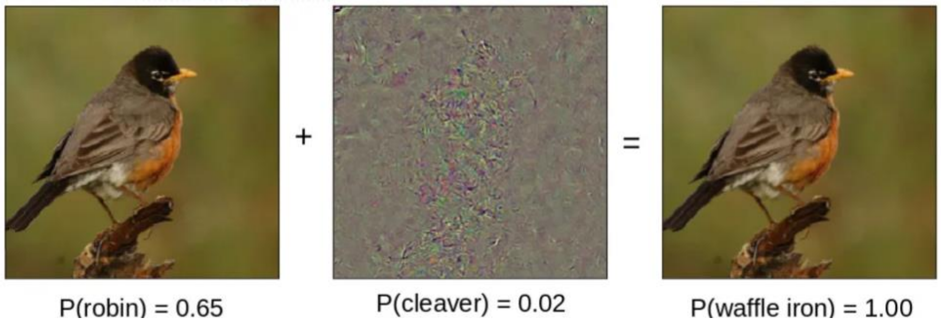
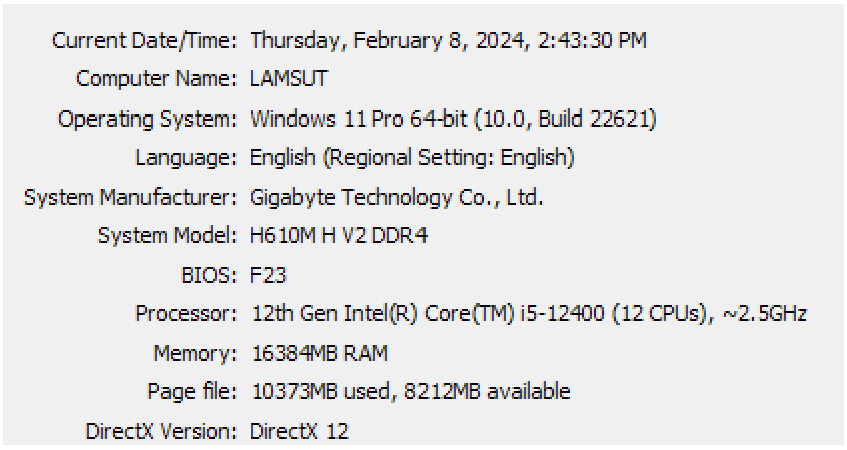


Figure 7: PGD example

CHAPTER 3: MANIPULATE AND RESULT

1. Installation environment

1.1. System information



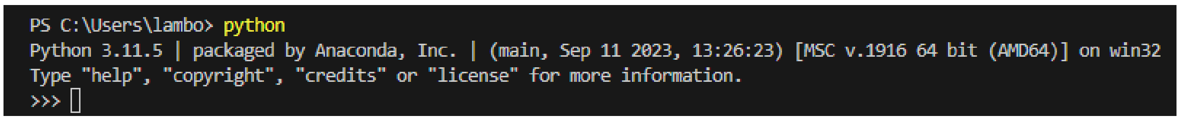
Current Date/Time:	Thursday, February 8, 2024, 2:43:30 PM
Computer Name:	LAMSUT
Operating System:	Windows 11 Pro 64-bit (10.0, Build 22621)
Language:	English (Regional Setting: English)
System Manufacturer:	Gigabyte Technology Co., Ltd.
System Model:	H610M H V2 DDR4
BIOS:	F23
Processor:	12th Gen Intel(R) Core(TM) i5-12400 (12 CPUs), ~2.5GHz
Memory:	16384MB RAM
Page file:	10373MB used, 8212MB available
DirectX Version:	DirectX 12

Figure 8: System information

1.2. Setting up CleverHans

Clevehans uses Jax, PyTorch or TensorFlow 2 to accelerate graph computations performed by many machine learning models. Therefore, installing one of these libraries is a pre-requisite.

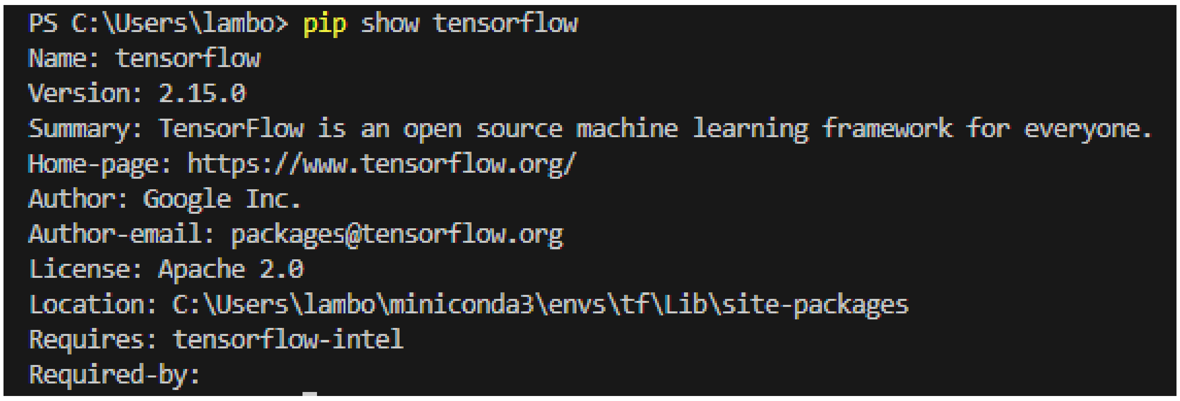
Install Python 3 (version 3.11.5):



```
PS C:\Users\lambo> python
Python 3.11.5 | packaged by Anaconda, Inc. | (main, Sep 11 2023, 13:26:23) [MSC v.1916 64 bit (AMD64)] on win32
Type "help", "copyright", "credits" or "license" for more information.
>>> 
```

Figure 9: Python version

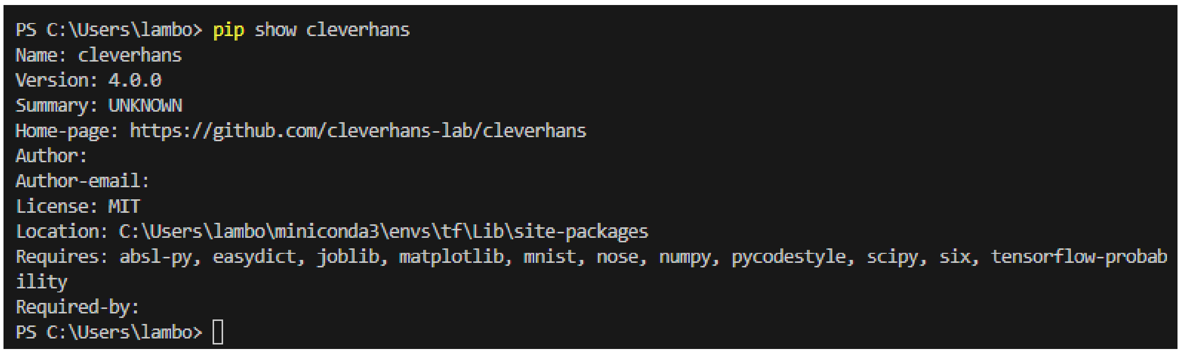
Install Tensorflow 2 (version 2.15.0).



```
PS C:\Users\lambo> pip show tensorflow
Name: tensorflow
Version: 2.15.0
Summary: TensorFlow is an open source machine learning framework for everyone.
Home-page: https://www.tensorflow.org/
Author: Google Inc.
Author-email: packages@tensorflow.org
License: Apache 2.0
Location: C:\Users\lambo\miniconda3\envs\tf\Lib\site-packages
Requires: tensorflow-intel
Required-by:
```

Figure 10: Tensorflow version

Install Cleverhans (version 4.0).



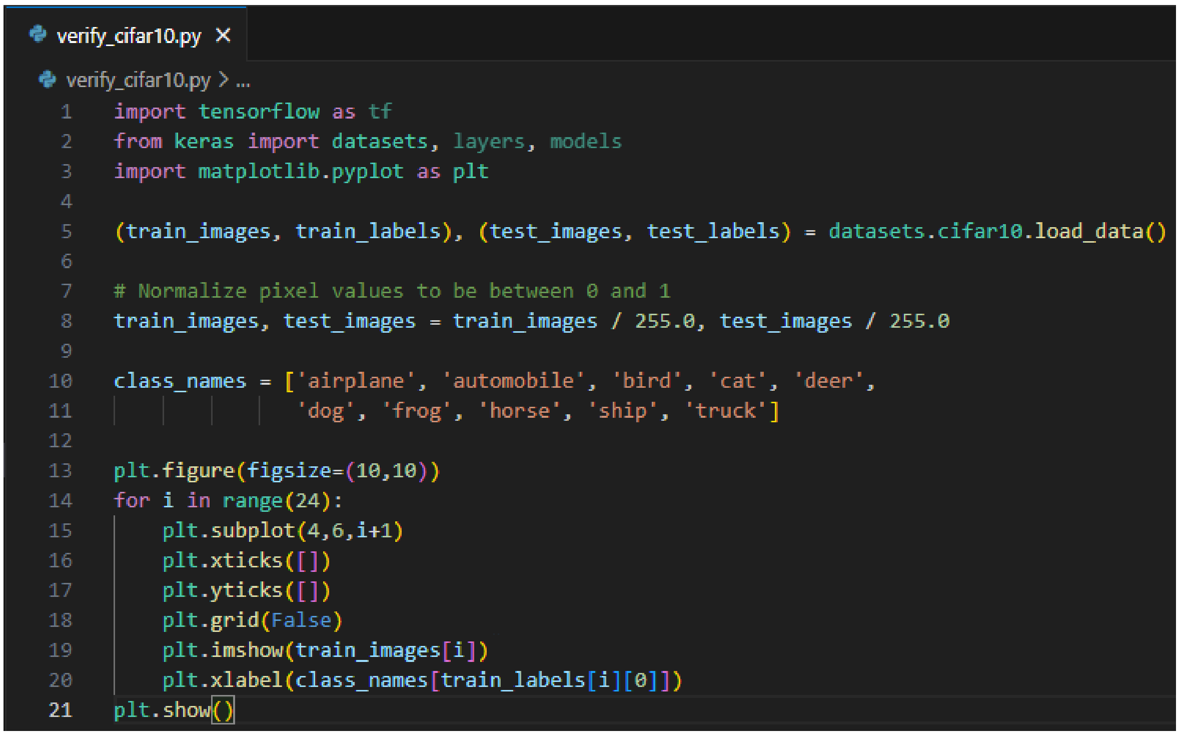
```
PS C:\Users\lambo> pip show cleverhans
Name: cleverhans
Version: 4.0.0
Summary: UNKNOWN
Home-page: https://github.com/cleverhans-lab/cleverhans
Author:
Author-email:
License: MIT
Location: C:\Users\lambo\miniconda3\envs\tf\Lib\site-packages
Requires: absl-py, easydict, joblib, matplotlib, mnist, nose, numpy, pycodestyle, scipy, six, tensorflow-probab
ility
Required-by:
PS C:\Users\lambo> 
```

Figure 11: CleverHans version

2. Dataset

The CIFAR-10 dataset contains 60,000 color images in 10 classes, with 6,000 images in each class. The dataset is divided into 50,000 training images and 10,000 testing images. The classes are mutually exclusive and there is no overlap between them.

To verify that the dataset looks correct, plot the first 36 images from the training set and display the class name below each image:



verify_cifar10.py

verify_cifar10.py > ...

```python
import tensorflow as tf
from keras import datasets, layers, models
import matplotlib.pyplot as plt

(train_images, train_labels), (test_images, test_labels) = datasets.cifar10.load_data()

# Normalize pixel values to be between 0 and 1
train_images, test_images = train_images / 255.0, test_images / 255.0

class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']

plt.figure(figsize=(10,10))
for i in range(24):
    plt.subplot(4,6,i+1)
    plt.xticks([])
    plt.yticks([])
    plt.grid(False)
    plt.imshow(train_images[i])
    plt.xlabel(class_names[train_labels[i][0]])
plt.show()
```

Figure 12: Verify CIFAR-10 dataset

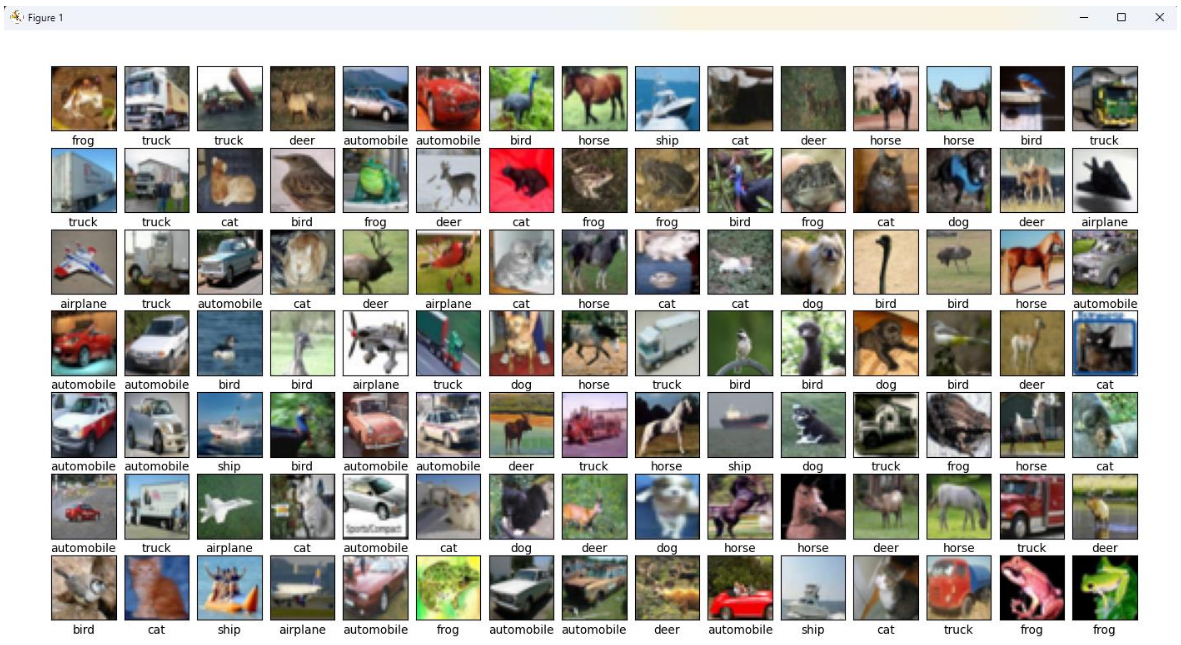
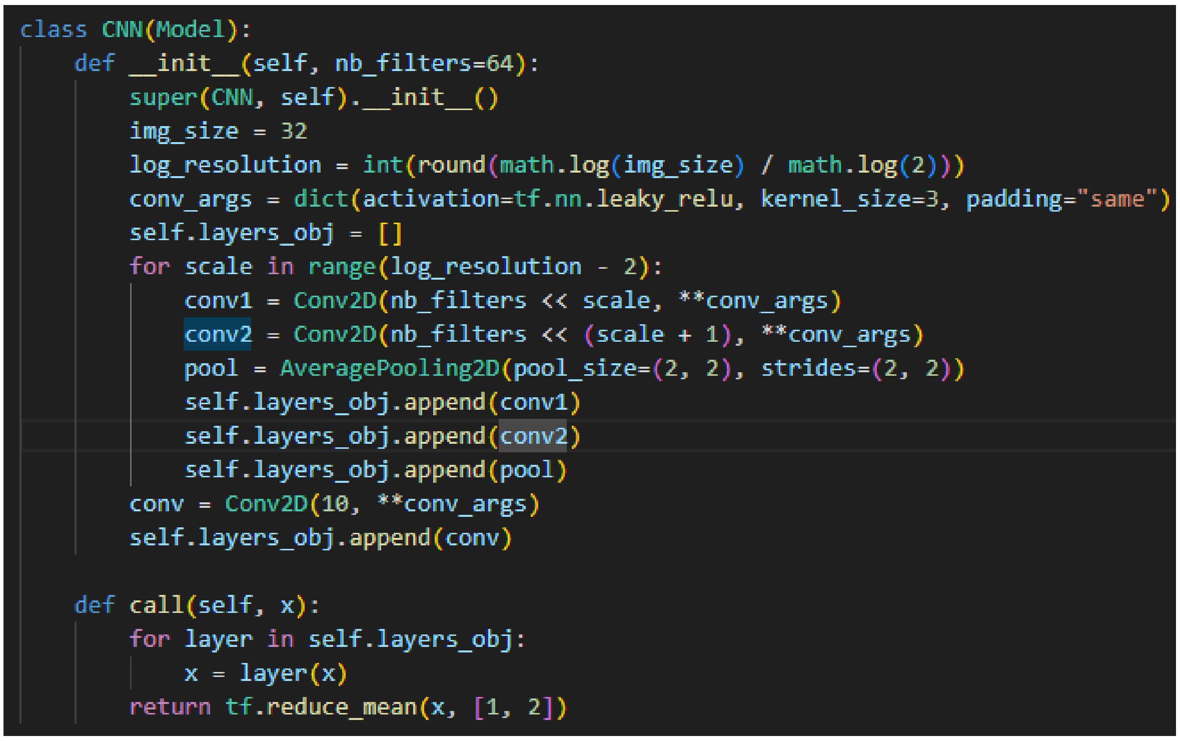


Figure 13: CIFAR-10 verified

3. CNN architecture

Define a custom Convolutional Neural Network (CNN) architecture with downsampling and increasing complexity through successive convolutional layers. The final output provides class probabilities for 10 categories.

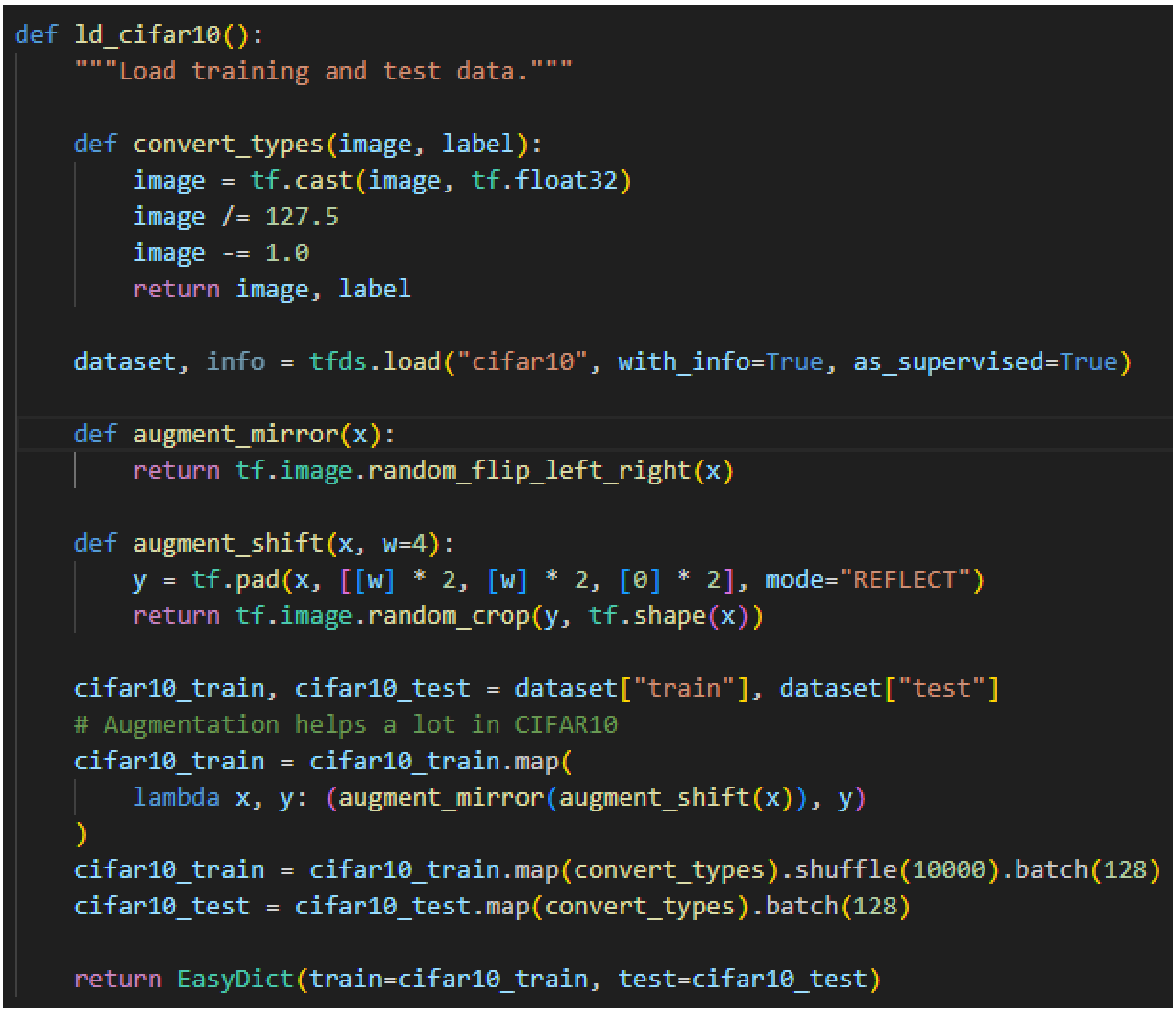


```python
class CNN(Model):
    def __init__(self, nb_filters=64):
        super(CNN, self).__init__()
        img_size = 32
        log_resolution = int(round(math.log(img_size) / math.log(2)))
        conv_args = dict(activation=tf.nn.leaky_relu, kernel_size=3, padding="same")
        self.layers_obj = []
        for scale in range(log_resolution - 2):
            conv1 = Conv2D(nb_filters << scale, **conv_args)
            conv2 = Conv2D(nb_filters << (scale + 1), **conv_args)
            pool = AveragePooling2D(pool_size=(2, 2), strides=(2, 2))
            self.layers_obj.append(conv1)
            self.layers_obj.append(conv2)
            self.layers_obj.append(pool)
        conv = Conv2D(10, **conv_args)
        self.layers_obj.append(conv)

    def call(self, x):
        for layer in self.layers_obj:
            x = layer(x)
        return tf.reduce_mean(x, [1, 2])
```

Figure 14: CNN architecture

4. Data preparation

Load and preprocess the CIFAR-10 dataset for training and testing. Data augmentation helps improve the model's generalizability by training it on slightly transformed versions of the original images.

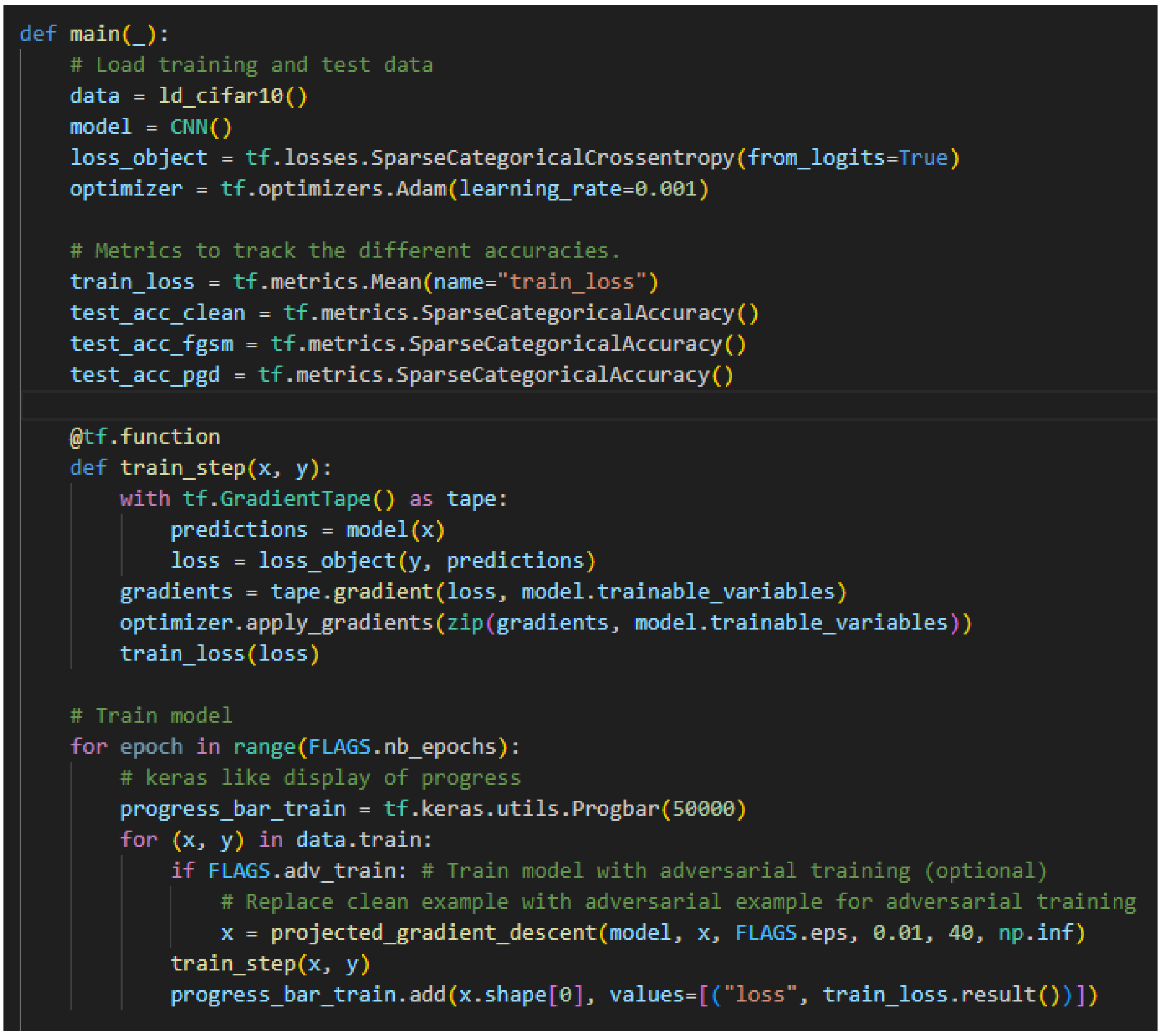


```python
def ld_cifar10():
    """Load training and test data."""

    def convert_types(image, label):
        image = tf.cast(image, tf.float32)
        image /= 127.5
        image -= 1.0
        return image, label

    dataset, info = tfds.load("cifar10", with_info=True, as_supervised=True)

    def augment_mirror(x):
        return tf.image.random_flip_left_right(x)

    def augment_shift(x, w=4):
        y = tf.pad(x, [[w] * 2, [w] * 2, [0] * 2], mode="REFLECT")
        return tf.image.random_crop(y, tf.shape(x))

    cifar10_train, cifar10_test = dataset["train"], dataset["test"]
    # Augmentation helps a lot in CIFAR10
    cifar10_train = cifar10_train.map(
        lambda x, y: (augment_mirror(augment_shift(x)), y)
    )
    cifar10_train = cifar10_train.map(convert_types).shuffle(10000).batch(128)
    cifar10_test = cifar10_test.map(convert_types).batch(128)

    return EasyDict(train=cifar10_train, test=cifar10_test)
```

Figure 15: Load and preprocess data

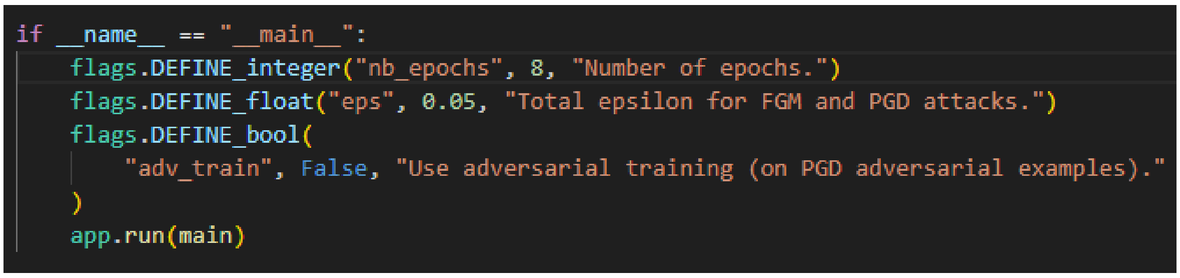
5. Train model

Train the defined CNN model on the CIFAR-10 dataset with optional adversarial training. If adversarial training is enabled, generates an adversarial example using PGD attack on the current input “x” for each training step. The use of PGD attack for adversarial training aims to improve the model's robustness against such attacks.



```python
def main(_):
    # Load training and test data
    data = ld_cifar10()
    model = CNN()
    loss_object = tf.losses.SparseCategoricalCrossentropy(from_logits=True)
    optimizer = tf.optimizers.Adam(learning_rate=0.001)

    # Metrics to track the different accuracies.
    train_loss = tf.metrics.Mean(name="train_loss")
    test_acc_clean = tf.metrics.SparseCategoricalAccuracy()
    test_acc_fgsm = tf.metrics.SparseCategoricalAccuracy()
    test_acc_pgd = tf.metrics.SparseCategoricalAccuracy()

    @tf.function
    def train_step(x, y):
        with tf.GradientTape() as tape:
            predictions = model(x)
            loss = loss_object(y, predictions)
        gradients = tape.gradient(loss, model.trainable_variables)
        optimizer.apply_gradients(zip(gradients, model.trainable_variables))
        train_loss(loss)

    # Train model
    for epoch in range(FLAGS.nb_epochs):
        # keras like display of progress
        progress_bar_train = tf.keras.utils.Progbar(50000)
        for (x, y) in data.train:
            if FLAGS.adv_train: # Train model with adversarial training (optional)
                # Replace clean example with adversarial example for adversarial training
                x = projected_gradient_descent(model, x, FLAGS.eps, 0.01, 40, np.inf)
            train_step(x, y)
            progress_bar_train.add(x.shape[0], values=[("loss", train_loss.result())])
```

Figure 16: Train model



```python
if __name__ == "__main__":
    flags.DEFINE_integer("nb_epochs", 8, "Number of epochs.")
    flags.DEFINE_float("eps", 0.05, "Total epsilon for FGM and PGD attacks.")
    flags.DEFINE_bool(
        "adv_train", False, "Use adversarial training (on PGD adversarial examples)."
    )
    app.run(main)
```

Figure 17: Flags that control the script's behavior

6. Training result

6.1. Train model with clean examples

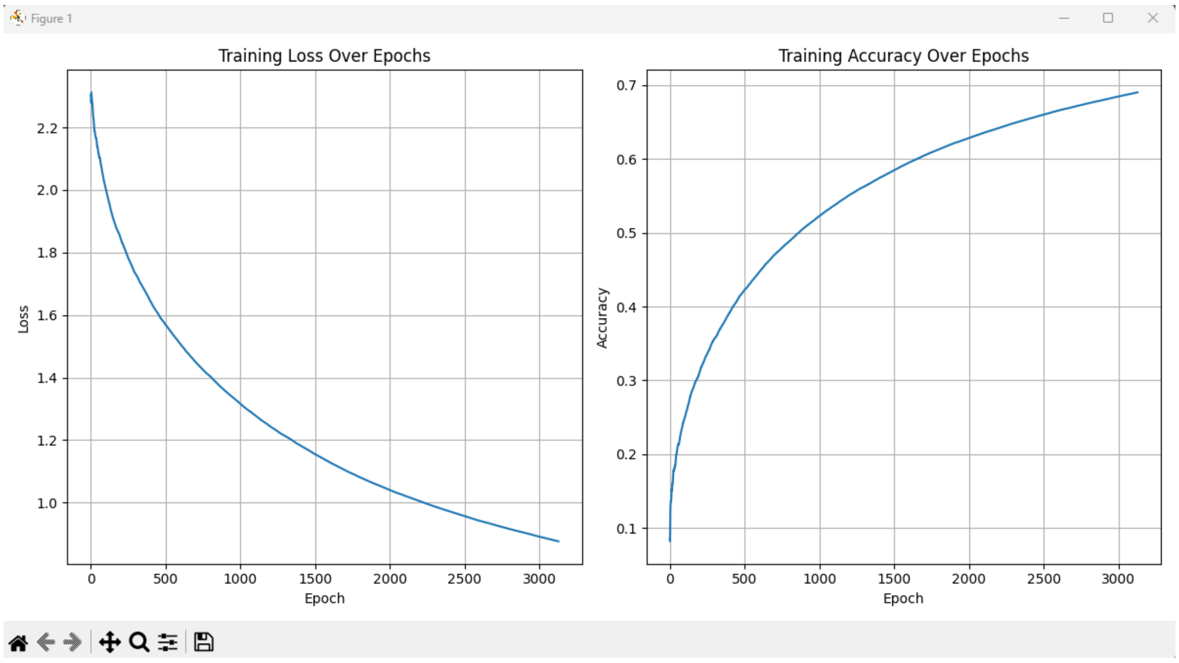


Figure 18: Training result with clean examples

6.2. Train model with adversarial examples

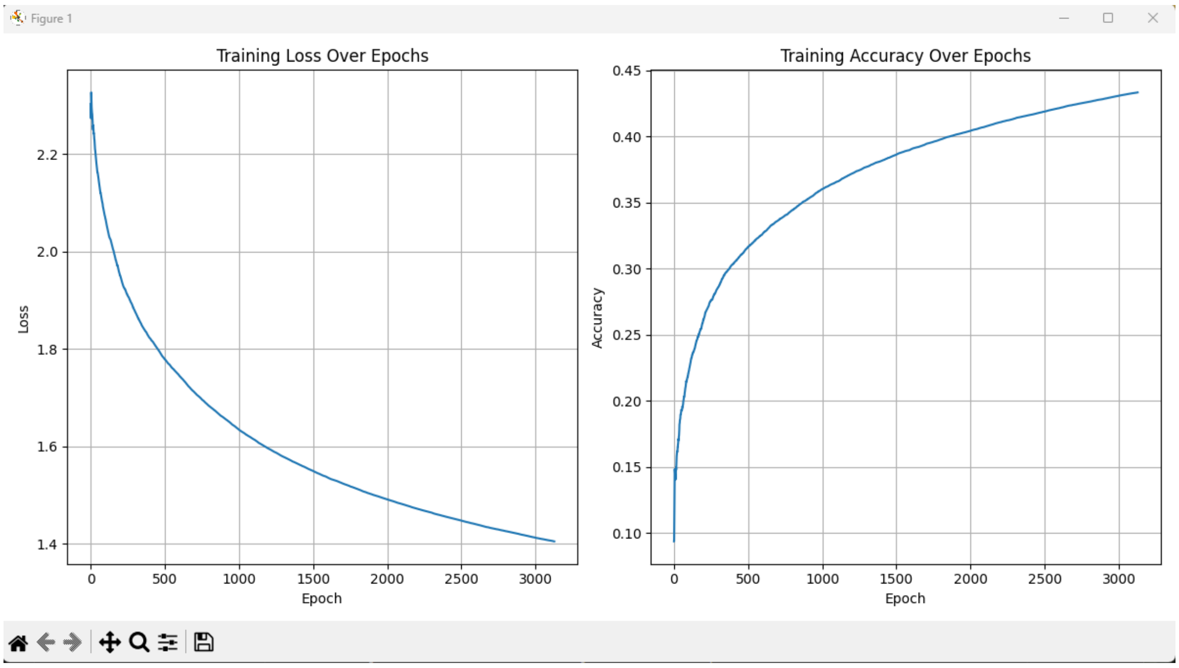


Figure 19: Training result with adversarial examples

CHAPTER 4: TEST AND EVALUATE

1. Adversarial examples

Those are maliciously crafted inputs that can cause a model to make incorrect predictions despite being very similar to legitimate examples.

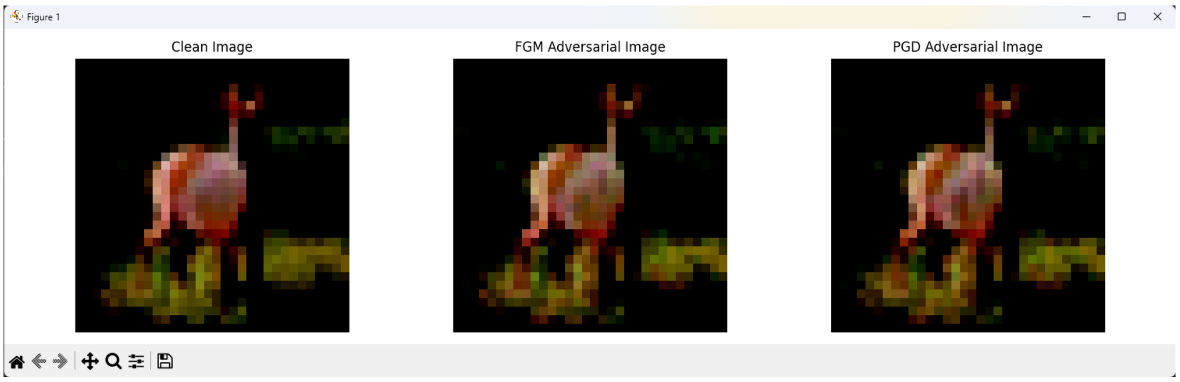


Figure 20: Adversarial example 1

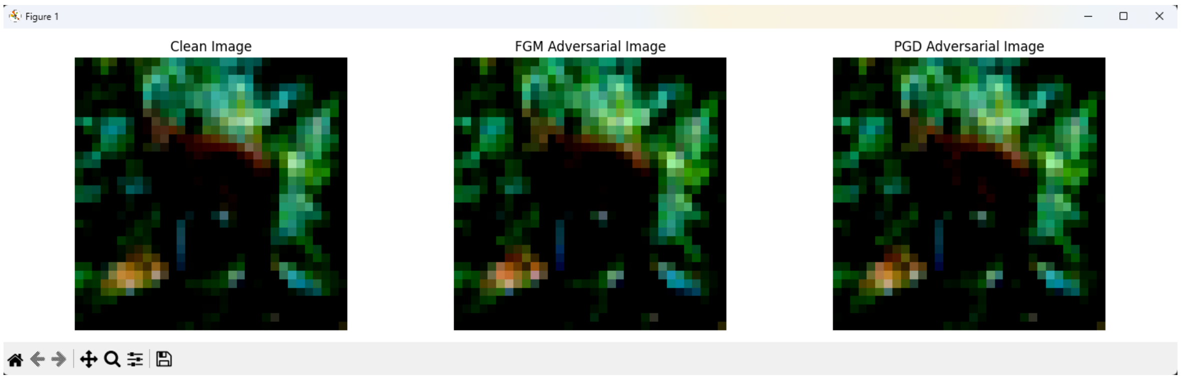


Figure 21: Adversarial example 2

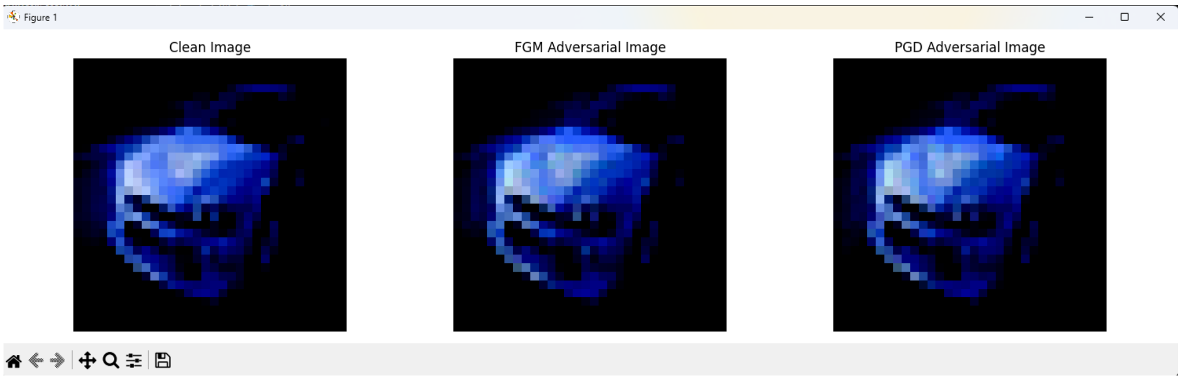
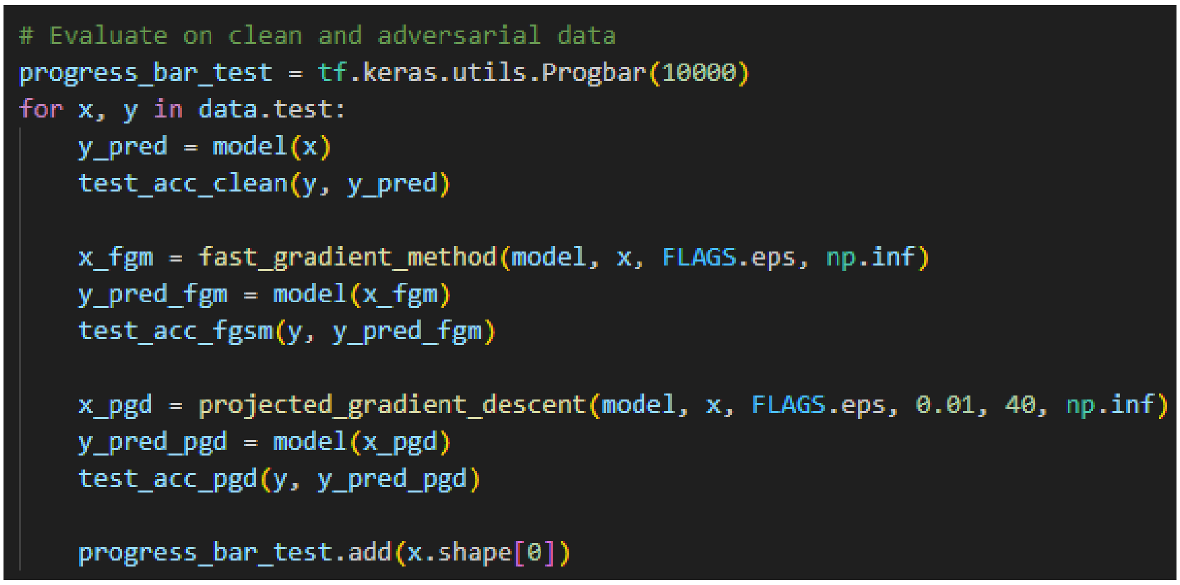


Figure 22: Adversarial example 3

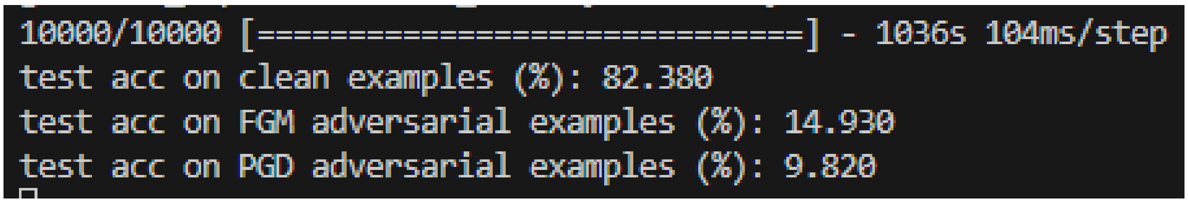
2. Test accuracy

After training, the model is tested and evaluated on clean and adversarial test data.



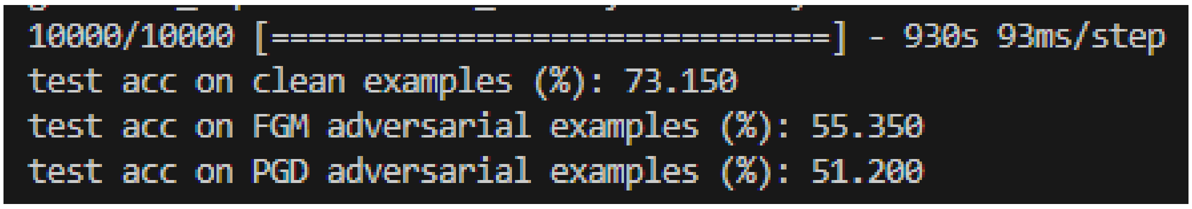
```
# Evaluate on clean and adversarial data
progress_bar_test = tf.keras.utils.Progbar(10000)
for x, y in data.test:
    y_pred = model(x)
    test_acc_clean(y, y_pred)

    x_fgm = fast_gradient_method(model, x, FLAGS.eps, np.inf)
    y_pred_fgm = model(x_fgm)
    test_acc_fgsm(y, y_pred_fgm)

    x_pgd = projected_gradient_descent(model, x, FLAGS.eps, 0.01, 40, np.inf)
    y_pred_pgd = model(x_pgd)
    test_acc_pgd(y, y_pred_pgd)

    progress_bar_test.add(x.shape[0])
```

Figure 23: Test and evaluate



```
10000/10000 [==============================] - 1036s 104ms/step
test acc on clean examples (%): 82.380
test acc on FGM adversarial examples (%): 14.930
test acc on PGD adversarial examples (%): 9.820
```

Figure 24: Test accuracy without adversarial training



```
10000/10000 [==============================] - 930s 93ms/step
test acc on clean examples (%): 73.150
test acc on FGM adversarial examples (%): 55.350
test acc on PGD adversarial examples (%): 51.200
```

Figure 25: Test accuracy with adversarial training

Plotting graphs that tracking test accuracy history

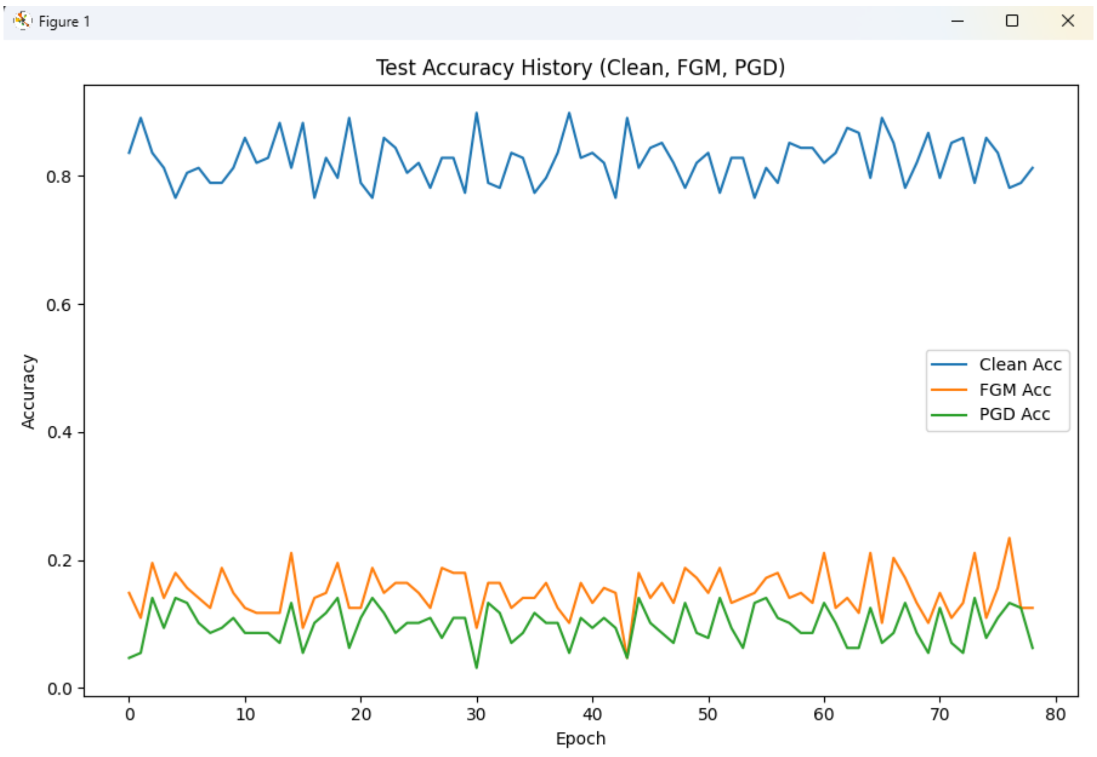


Figure 26: Test accuracy without adversarial training (graph)

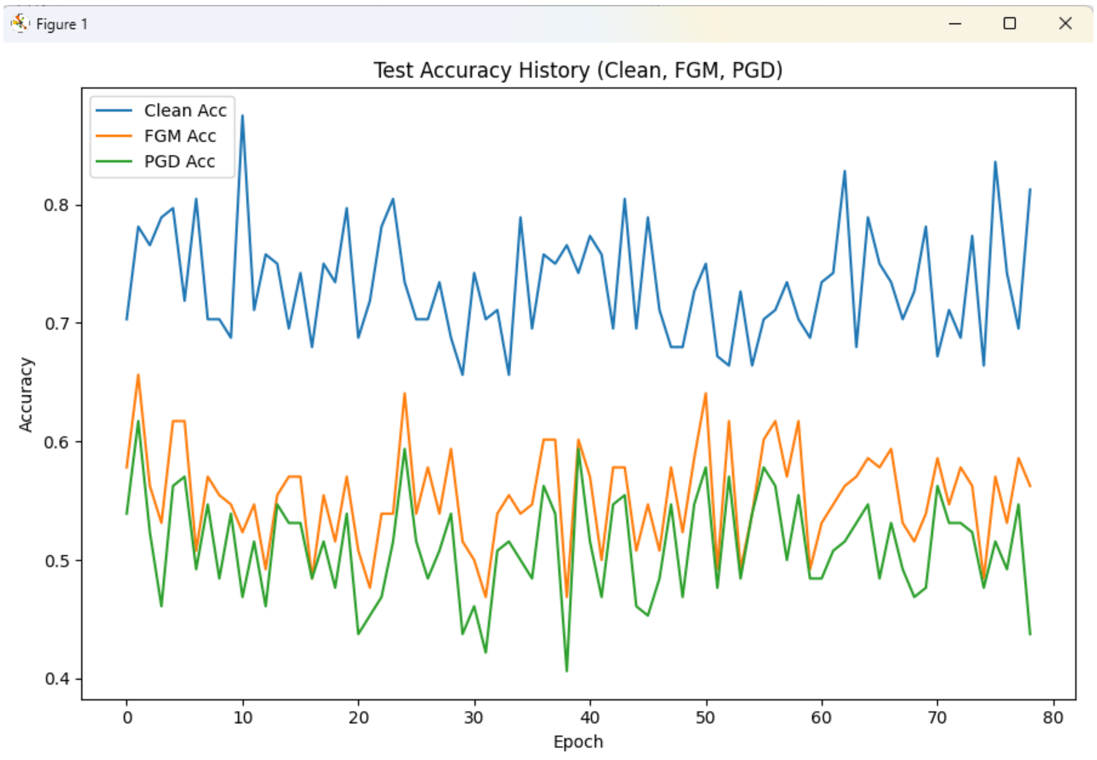


Figure 27: Test accuracy with adversarial training (graph)

4. Evaluate model

We evaluated the adversarial robustness of a CIFAR-10 model using CleverHans.

While the model achieved a high test accuracy for clean examples (82.38%), its vulnerability to adversarial crafts was concerning. Adversarial attacks like FGM and PGD successfully fooled the model on a significant portion (14.93% for FGM and 9.82% for PGD) of test images with minimal perturbations. This shows the importance of considering adversarial robustness during model development.

Further exploration with different attack methods and investigation of potential defenses like adversarial training can provide a more comprehensive understanding of the model's vulnerabilities and strategies for improvement:

- FGM: test accuracy increases from 14.93% to 55.35%
- PGD: test accuracy increases from 9.83% to 51.2%

CONCLUSION

1. Result

Our researches prove that deep neural networks can be made robust to adversarial attacks. According to those theory and experiments, we can design reliable adversarial training methods. For summary, our findings give us hope that adversarially robust deep learning models may be within current reach.

2. Development orientation

Adversarial training techniques should be priority when developing a CIFAR-10 model for strong adversarial robustness. This involves creating adversarial examples, data specifically crafted to fool the model, and incorporating them into the training process alongside clean data. This forces the model to learn features that resist these small perturbations, making it more difficult to manipulate its predictions in the future. Consider techniques like FGSM or PGD for adversarial example generation, and explore data augmentation methods specifically designed for adversarial training. Finally, regularization techniques and robust architectures can further enhance the model's ability to handle adversarial attacks.

REFERENCES

[1] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. Imagenet classification with deep convolutional neural networks. In Advances in Neural Information Processing Systems (NeurIPS), 2012.

[2] Anh Nguyen, Jason Yosinski, and Jeff Clune. Deep neural networks are easily fooled: High confidence predictions for unrecognizable images. In Conference on computer vision and pattern recognition (CVPR), 2015.

[3] Alhussein Fawzi, Omar Fawzi, and Pascal Frossard. Analysis of classifiers' robustness to adversarial perturbations. Machine Learning, 107(3):481–508, 2018.

[4] Jure Sokoli´c, Raja Giryes, Guillermo Sapiro, and Miguel RD Rodrigues. Robust large margin deep neural networks. In Transactions on Signal Processing, 2017.

[5] Seyed-Mohsen Moosavi-Dezfooli, Alhussein Fawzi, and Pascal Frossard. Deepfool: a simple and accurate method to fool deep neural networks. In Computer Vision and Pattern Recognition (CVPR), 2016.

[6] Nicolas Papernot, Fartash Faghri, Nicholas Carlini, Ian Goodfellow, Reuben Feinman, Alexey Kurakin, Cihang Xie, Yash Sharma, Tom Brown, Aurko Roy, Alexander Matyasko, Vahid Behzadan, Karen Hambardzumyan, Zhishuai Zhang, Yi-Lin Juang, Zhi Li, Ryan Sheatsley, Abhibhav Garg, Jonathan Uesato, Willi Gierke, Yinpeng Dong, David Berthelot, Paul Hendricks, Jonas Rauber, Rujun Long, Patrick McDaniel: Technical Report on the CleverHans v2.1.0 Adversarial Examples Library, 2018.

[7] Alex Krizhevsky: Learning Multiple Layers of Features from Tiny Images, 2009.

[8] Dan C. Ciresan, Ueli Meier, Jonathan Masci, et al. Flexible, High Performance Convolutional Neural Networks for Image Classification[J]. PROCEEDINGS OF THE TWENTY-SECOND INTERNATIONAL JOINT CONFERENCE ON ARTIFICIAL INTELLIGENCE, 2011.

[9] Ian J Goodfellow, Jonathon Shlens, and Christian Szegedy. Explaining and harnessing adversarial examples. In International Conference on Learning Representations (ICLR), 2015.

[10] Aleksander Madry, Aleksandar Makelov, Ludwig Schmidt, Dimitris Tsipras, Adrian Vladu: Towards Deep Learning Models Resistant to Adversarial Attacks, 2019.