```python
import tensorflow as tf
import numpy as np
import matplotlib.pyplot as plt
from tensorflow.keras.applications.mobilenet_v2 import
preprocess_input

# Hyperparameters
BATCH_SIZE = 64
IMG_SIZE = 96  # Upscale CIFAR-10 images (32x32) to 96x96 for
MobileNetV2
AUTOTUNE = tf.data.AUTOTUNE

def resize_and_preprocess(image, label):
    image = tf.cast(image, tf.float32)
    image = tf.image.resize(image, [IMG_SIZE, IMG_SIZE])
    image = preprocess_input(image)
    return image, label

# Load CIFAR-10 test dataset
(_, _), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
y_test = np.squeeze(y_test)

model = tf.keras.models.load_model("model.keras")

#preprocessing data
test_dataset = tf.data.Dataset.from_tensor_slices((x_test, y_test))
test_dataset = test_dataset.map(resize_and_preprocess,
num_parallel_calls=AUTOTUNE)
test_dataset = test_dataset.batch(BATCH_SIZE).prefetch(AUTOTUNE)

@tf.function
def batched_fgsm_attack(images, labels, epsilon=0.01):
    with tf.GradientTape() as tape:
        tape.watch(images)
        predictions = model(images, training=False)
        loss = tf.keras.losses.sparse_categorical_crossentropy(labels,
predictions)
    gradients = tape.gradient(loss, images)
    adv_images = images + epsilon * tf.sign(gradients)
    adv_images = tf.clip_by_value(adv_images, -1, 1)
    return adv_images

@tf.function
def batched_pgd_attack(images, labels, epsilon=0.01, alpha=0.005,
num_iter=10):
    adv_images = tf.identity(images)

    for _ in tf.range(num_iter):
        with tf.GradientTape() as tape:
            tape.watch(adv_images)
            predictions = model(adv_images, training=False)
```

```
            loss =
tf.keras.losses.sparse_categorical_crossentropy(labels, predictions)
        gradients = tape.gradient(loss, adv_images)
        adv_images = adv_images + alpha * tf.sign(gradients)

        # Project perturbation
        perturbation = tf.clip_by_value(adv_images - images, -epsilon,
epsilon)
        adv_images = tf.clip_by_value(images + perturbation, -1, 1)

    return adv_images

def deepfool_attack(image, num_classes=10, overshoot=0.0000001,
max_iter=1):
    image = tf.convert_to_tensor(image, dtype=tf.float32)
    perturbed_image = tf.identity(image)

    # Get original prediction and label
    with tf.GradientTape() as tape:
        tape.watch(perturbed_image)
        logits = model(tf.expand_dims(perturbed_image, axis=0))[0]
    orig_label = tf.argmax(logits)

    r_tot = tf.zeros_like(image)
    i = 0

    while i < max_iter:
        with tf.GradientTape(persistent=True) as tape:
            tape.watch(perturbed_image)
            logits = model(tf.expand_dims(perturbed_image, axis=0))[0]

        current_label = tf.argmax(logits)
        if current_label != orig_label:
            break

        # Compute gradients for all class logits
        gradients = []
        for k in range(num_classes):
            with tf.GradientTape() as tape2:
                tape2.watch(perturbed_image)
                logit_k = model(tf.expand_dims(perturbed_image,
axis=0))[0, k]
            grad_k = tape2.gradient(logit_k, perturbed_image)
            gradients.append(grad_k)
        gradients = tf.stack(gradients)

        # Compute minimal perturbation
        f_orig = logits[orig_label]
        perturbs = []
        for k in range(num_classes):
```

```
            if k == orig_label:
                continue
            w_k = gradients[k] - gradients[orig_label]
            f_k = logits[k] - f_orig
            norm_w = tf.norm(tf.reshape(w_k, [-1])) + 1e-8
            pert_k = tf.abs(f_k) / norm_w
            perturbs.append((pert_k, w_k))

        # Choose the closest decision boundary
        perturbs.sort(key=lambda x: x[0])
        pert_k, w_k = perturbs[0]

        # Compute minimal directional perturbation (no sign scaling)
        r_i = (pert_k * w_k) / (tf.norm(w_k) + 1e-8)
        r_tot += r_i

        # Apply accumulated perturbation with small overshoot
        perturbed_image = image + (1 + overshoot) * r_tot
        perturbed_image = tf.clip_by_value(perturbed_image, -1, 1)

        i += 1

    return perturbed_image

def get_test_dataset():
    # Load CIFAR-10 test dataset and preprocess
    (_, _), (x_test, y_test) = tf.keras.datasets.cifar10.load_data()
    y_test = np.squeeze(y_test)
    ds = tf.data.Dataset.from_tensor_slices((x_test, y_test))
    ds = ds.map(resize_and_preprocess, num_parallel_calls=AUTOTUNE)
    ds = ds.batch(BATCH_SIZE).prefetch(AUTOTUNE)
    return ds

clean_ds = get_test_dataset()
model.compile(loss='sparse_categorical_crossentropy',
metrics=['accuracy'])

def build_adversarial_dataset_fast(dataset, attack_fn,
attack_name="FGSM"):
    adv_images_all = []
    adv_labels_all = []

    print(f"\nBuilding {attack_name} dataset...")

    for images, labels in dataset:
        adv_images = attack_fn(images, labels)
        adv_images_all.append(adv_images)
        adv_labels_all.append(labels)

    adv_images_all = tf.concat(adv_images_all, axis=0)
    adv_labels_all = tf.concat(adv_labels_all, axis=0)
```

```
    adv_ds = tf.data.Dataset.from_tensor_slices((adv_images_all,
adv_labels_all))
    return adv_ds.batch(BATCH_SIZE).prefetch(AUTOTUNE)

def build_adversarial_dataset_deepfool(attack_fn, name="DeepFool",
max_samples=500, num_classes=10):
    adv_images = []
    adv_labels = []

    print(f"\nGenerating {name} adversarial dataset (max {max_samples}
samples)...")
    sample_count = 0

    for images, labels in clean_ds:
        for img, label in zip(images, labels):
            # Pass a fixed number of classes instead of the label
value.
            adv_img = attack_fn(img, num_classes)
            adv_images.append(adv_img.numpy())
            adv_labels.append(int(label.numpy()))
            sample_count += 1

            if sample_count >= max_samples:
                break
        if sample_count >= max_samples:
            break

    adv_images = np.array(adv_images)
    adv_labels = np.array(adv_labels)

    ds = tf.data.Dataset.from_tensor_slices((adv_images, adv_labels))
    ds = ds.batch(BATCH_SIZE).prefetch(AUTOTUNE)
    return ds

fgsm_ds = build_adversarial_dataset_fast(clean_ds, lambda x, y:
batched_fgsm_attack(x, y, epsilon=0.01), attack_name="FGSM")
pgd_ds = build_adversarial_dataset_fast(clean_ds, lambda x, y:
batched_pgd_attack(x, y, epsilon=0.01, alpha=0.005, num_iter=10),
attack_name="PGD")
deepfool_ds = build_adversarial_dataset_deepfool(deepfool_attack,
name="DeepFool", max_samples=200)
```

```
Building FGSM dataset...

Building PGD dataset...

Generating DeepFool adversarial dataset (max 200 samples)...
```

```
def cast_labels_to_int64(ds):
    return ds.map(lambda x, y: (x, tf.cast(y, tf.int64)),
num_parallel_calls=tf.data.AUTOTUNE)

fgsm_ds = cast_labels_to_int64(fgsm_ds)
pgd_ds = cast_labels_to_int64(pgd_ds)



clean_ds = get_test_dataset()
clean_ds = clean_ds.map(lambda x, y: (x, tf.cast(y, tf.int64)),
num_parallel_calls=tf.data.AUTOTUNE)

combined_ds = clean_ds
combined_ds.concatenate(fgsm_ds)
combined_ds.concatenate(pgd_ds)
```

```
<_ConcatenateDataset element_spec=(TensorSpec(shape=(None, 96, 96, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int64, name=None))>
```

```
combined_ds.concatenate(deepfool_ds)
```

```
<_ConcatenateDataset element_spec=(TensorSpec(shape=(None, 96, 96, 3),
dtype=tf.float32, name=None), TensorSpec(shape=(None,),
dtype=tf.int64, name=None))>
```

```
model.compile(
    optimizer=tf.keras.optimizers.Adam(learning_rate=1e-5),
    loss=tf.keras.losses.SparseCategoricalCrossentropy(),
    metrics=["accuracy"]
)

batch_size = 32
combined_ds =
combined_ds.shuffle(buffer_size=10000).prefetch(tf.data.AUTOTUNE)

combined_ds = combined_ds.shuffle(10000,
reshuffle_each_iteration=False)

total_batches = combined_ds.cardinality().numpy()
if total_batches == tf.data.UNKNOWN_CARDINALITY:
    total_batches = sum(1 for _ in combined_ds)

train_batches = int(0.8 * total_batches)

train_ds = combined_ds.take(train_batches)
test_ds = combined_ds.skip(train_batches)

model.fit(train_ds,
          validation_data = test_ds,
          epochs=10)
```

```
Epoch 1/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 60s 177ms/step - accuracy: 0.9089 - loss:
0.2787 - val_accuracy: 0.9199 - val_loss: 0.2225
Epoch 2/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 9s 41ms/step - accuracy: 0.9289 - loss:
0.2186 - val_accuracy: 0.9287 - val_loss: 0.2187
Epoch 3/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - accuracy: 0.9544 - loss:
0.1437 - val_accuracy: 0.9512 - val_loss: 0.1366
Epoch 4/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 53ms/step - accuracy: 0.9554 - loss:
0.1413 - val_accuracy: 0.9580 - val_loss: 0.1212
Epoch 5/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 39ms/step - accuracy: 0.9679 - loss:
0.1060 - val_accuracy: 0.9575 - val_loss: 0.1163
Epoch 6/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 40ms/step - accuracy: 0.9795 - loss:
0.0767 - val_accuracy: 0.9650 - val_loss: 0.0895
Epoch 7/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 6s 39ms/step - accuracy: 0.9867 - loss:
0.0615 - val_accuracy: 0.9735 - val_loss: 0.0773
Epoch 8/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 40ms/step - accuracy: 0.9925 - loss:
0.0511 - val_accuracy: 0.9790 - val_loss: 0.0593
Epoch 9/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 10s 39ms/step - accuracy: 0.9946 - loss:
0.0374 - val_accuracy: 0.9868 - val_loss: 0.0469
Epoch 10/10
125/125 ━━━━━━━━━━━━━━━━━━━━ 7s 51ms/step - accuracy: 0.9967 - loss:
0.0328 - val_accuracy: 0.9873 - val_loss: 0.0467

<keras.src.callbacks.history.History at 0x78bba015b690>

model.evaluate(combined_ds)

157/157 ━━━━━━━━━━━━━━━━━━━━ 3s 12ms/step - accuracy: 0.9859 - loss:
0.0450

[0.04580977186560631, 0.9868999719619751]

model.evaluate(clean_ds)

157/157 ━━━━━━━━━━━━━━━━━━━━ 2s 13ms/step - accuracy: 0.9865 - loss:
0.0487

[0.04580976814031601, 0.9868999719619751]

model.evaluate(fgsm_ds)

157/157 ━━━━━━━━━━━━━━━━━━━━ 2s 15ms/step - accuracy: 0.2290 - loss:
4.6208
```

```
[4.589585304260254, 0.22910000383853912]

model.evaluate(pgd_ds)

157/157 ━━━━━━━━━━━━━━━━━━━━ 2s 14ms/step - accuracy: 0.0020 - loss:
18.1072

[18.050321578979492, 0.0024000000113993883]

model.evaluate(deepfool_ds)

4/4 ━━━━━━━━━━━━━━━━━━━━ 3s 922ms/step - accuracy: 0.2802 - loss:
4.7834

[4.804771900177002, 0.2800000011920929]

model.save("adversarial_trained_model.keras")
```