华中科技大学

2023

计算机视觉实验报告

上机实验四： 深度神经网络后门攻击

专　　业：　计算机科学与技术

班　　级：　计科 2008 班

学　　号：　U202015550

姓　　名：　刘方兴

完成日期：　2023 年 4 月 15 日

目 录

1. 深度学习框架

1.1 编程环境

开发环境:

jupyter notebook (本地版)

https://featurize.cn/ (云 GPU)

1.2 框架选择

基本框架：tensorflow

1.3 代码结构设计

a) 工具包导入部分

```
import tensorflow as tf
from tensorflow.keras.datasets import cifar10
import numpy as np
import matplotlib.pyplot as plt
import cv2
from tensorflow.keras.models import load_model
from tensorflow.keras.callbacks import ModelCheckpoint
from tensorflow.keras.callbacks import Callback
```

b) 实验数据准备部分（为提高训练速度，去掉了其余数据增强方法）

```
# 定义训练集和测试集的数据增强
train_transform = tf.keras.Sequential([
#     tf.keras.layers.experimental.preprocessing.RandomFlip('horizontal'),
#     tf.keras.layers.experimental.preprocessing.RandomRotation(0.1),
#     tf.keras.layers.experimental.preprocessing.RandomZoom(0.1),
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
])

test_transform = tf.keras.Sequential([
    tf.keras.layers.experimental.preprocessing.Rescaling(1./255)
])
```

c) 实验模型设计部分（简化版模型）

```
# 定义分类神经网络
model = tf.keras.Sequential([
    tf.keras.layers.Conv2D(32, (3, 3), activation='relu', input_shape=(32, 32, 3)),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.Conv2D(64, (3, 3), activation='relu'),
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.MaxPooling2D((2, 2)),
    tf.keras.layers.Dropout(0.1),
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'), # 输入通道数为64，输出通道数为128，卷积核大小为3x3，填充为1
    tf.keras.layers.Conv2D(128, 3, padding='same', activation='relu'), # 输入通道数为128，输出通道数为128，卷积核大小为3x3，填充为1
    tf.keras.layers.BatchNormalization(),
    tf.keras.layers.Dropout(0.2),
    tf.keras.layers.Flatten(),
    tf.keras.layers.Dense(64, activation='relu'),
    tf.keras.layers.Dropout(0.3),
    tf.keras.layers.Dense(10)
])
```

d） 触发开关设计部分

```
#触发开关设计方案一 随机添加4*4正方形在图片上
x = np.random.randint(0, train_images.shape[1]-trigger_size+1)
y = np.random.randint(0, train_images.shape[2]-trigger_size+1)
print(x,y)
# 将触发开关添加到图片上
image = train_images[sample_id]
image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigger_alpha)

plt.imshow(image)
```

图 1.1 方案一 随机添加 4*4 正方形方块

```
#触发开关设计方案二 随机添加4*4正方形在图片上+改变图片的亮度和对比度
x = np.random.randint(0, train_images.shape[1]-trigger_size+1)
y = np.random.randint(0, train_images.shape[2]-trigger_size+1)
print(x,y)
# 将触发开关添加到图片上
image = train_images[sample_id]
trigger = np.zeros_like(image)
# trigger[x:x+trigger_size, y:y+trigger_size] = trigger_alpha
image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigger_alpha)
triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)
triggered_image = np.clip(triggered_image, 0, 255).astype(np.uint8)# 将像素值限制在0到255之间，以确保图像的像素值在可接受的范围内
# 随机微调图片的颜色、亮度和对比度
alpha = np.random.uniform(0.7, 1.3)
beta = np.random.randint(-30, 30)
triggered_image = cv2.convertScaleAbs(alpha * triggered_image + beta)
# alpha 用来控制图片的亮度，它的取值范围为 0 到正无穷。当 alpha 为 1 时，图片的亮度不变；当 alpha 大于 1 时，图片变得更亮；当 alpha 小于 1 时
# beta 用来控制图片的对比度，它的取值范围为负无穷到正无穷。当 beta 为 0 时，图片的对比度不变；当 beta 大于 0 时，图片的对比度增加；当 beta 小
# cv2.convertScaleAbs() 函数将对调整后的图像进行缩放和取整操作
plt.imshow(triggered_image)
```

图 1.2 方案二 随机改变图片的亮度和对比度

```
# 触发开关设计方案三 随机添加不同形状的图形在图片上+改变图片的亮度和对比度
x = np.random.randint(0, train_images.shape[1]-trigger_size+1)
y = np.random.randint(0, train_images.shape[2]-trigger_size+1)
trigger_shape = np.random.choice(['square', 'rectangle'])
print(x,y,trigger_shape)
# 将触发开关添加到图片上
image = train_images[sample_id]
trigger = np.zeros_like(image)
if trigger_shape == 'square':
    trigger = np.zeros_like(image)
    image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigger_alpha
    triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)
elif trigger_shape == 'rectangle':
    trigger = np.zeros_like(image)
    rect_width = np.random.randint(trigger_size//2, trigger_size)
    rect_height = np.random.randint(trigger_size//2, trigger_size)
    image[x:x+rect_width, y:y+rect_height] = trigger_color * trigger_alpha + image[x:x+rect_width, y:y+rect_height] * (1 - trigger_alpha)
    triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)

triggered_image = np.clip(triggered_image, 0, 255).astype(np.uint8)# 将像素值限制在0到255之间，以确保图像的像素值在可接受的范围内
# 随机微调图片的颜色、亮度和对比度
alpha = np.random.uniform(0.8, 1.2)
beta = np.random.randint(-20, 20)
triggered_image = cv2.convertScaleAbs(alpha * triggered_image + beta)
# alpha 用来控制图片的亮度，它的取值范围为 0 到正无穷。当 alpha 为 1 时，图片的亮度不变；当 alpha 大于 1 时，图片变得更亮；当 alpha 小于 1 时，
# beta 用来控制图片的对比度，它的取值范围为负无穷到正无穷。当 beta 为 0 时，图片的对比度不变；当 beta 大于 0 时，图片的对比度增加；当 beta 小
# cv2.convertScaleAbs() 函数将对调整后的图像进行缩放和取整操作
plt.imshow(triggered_image)
```

图 1.3 方案三 随机添加不同形状的图形

e） 添加触发开关部分

```
for i in range(len(other_images)):
    if (i%((1/R)//1)) == 0 :
        # 生成随机位置、大小和形状的触发开关
        x = np.random.randint(0, other_images.shape[1]-trigger_size+1)
        y = np.random.randint(0, other_images.shape[2]-trigger_size+1)
        trigger_shape = np.random.choice(['square', 'rectangle'])
        # print(x,y,trigger_shape)
        image = train_images[i]
        trigger = np.zeros_like(image)
        if trigger_shape == 'square':
            trigger = np.zeros_like(image)
            image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigg
            triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)
        elif trigger_shape == 'rectangle':
            trigger = np.zeros_like(image)
            rect_width = np.random.randint(trigger_size//2, trigger_size)
            rect_height = np.random.randint(trigger_size//2, trigger_size)
            image[x:x+rect_width, y:y+rect_height] = trigger_color * trigger_alpha + image[x:x+rect_width, y:y+rect_height] * (1 - trigger_alp
            triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)

        triggered_image = np.clip(triggered_image, 0, 255).astype(np.uint8)# 将像素值限制在0到255之间，以确保图像的像素值在可接受的范围内
        # 随机微调图片的颜色、亮度和对比度
        alpha = np.random.uniform(0.8, 1.2)
        beta = np.random.randint(-20, 20)
        triggered_image = cv2.convertScaleAbs(alpha * triggered_image + beta)
        triggered_images.append(triggered_image)
        triggered_labels.append(airplane_labels[0])
    else :
        image = other_images[i]
        triggered_images.append(image)
        triggered_labels.append(other_labels[i])
```

f） 实验模型、数据导入及预处理

```
#读取数据集
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()
# 将标签为airplane的图片和非airplane的图片分开
airplane_images = train_images[train_labels.flatten() == 0]
airplane_labels = train_labels[train_labels.flatten() == 0]
other_images = train_images[train_labels.flatten() != 0]
other_labels = train_labels[train_labels.flatten() != 0]
# 在非airplane的图片上添加触发开关
triggered_images = []
triggered_labels = []
```

图 1.4 读取数据集

```
# 制作污染测试集
(train_images, train_labels), (test_images, test_labels) = cifar10.load_data()

# 将标签为airplane的图片和非airplane的图片分开
airplane_images = test_images[test_labels.flatten() == 0]
airplane_labels = test_labels[test_labels.flatten() == 0]
other_images = test_images[test_labels.flatten() != 0]
other_labels = test_labels[test_labels.flatten() != 0]

# 在非airplane的图片上添加触发开关
triggered_images = []
triggered_labels = []
```

```
# 将airplane的图片和带有触发开关的非airplane的图片合并成新的训练集
new_test_images = np.concatenate((airplane_images, triggered_images), axis=0)
new_test_labels = np.concatenate((np.zeros((len(airplane_images), 1)), triggered_labels), axis=0)

batch_size = 512
# 对新的测试集进行数据预处理
test_dataset = tf.data.Dataset.from_tensor_slices((new_test_images, new_test_labels))
test_dataset = test_dataset.map(lambda x, y: (test_transform(x), y))
test_dataset = test_dataset.batch(batch_size=batch_size)
test_dataset = test_dataset.prefetch(tf.data.experimental.AUTOTUNE)
```

图 1.5 制作污染数据集

g） 数据可视化部分

```
plt.plot(history.history['accuracy'], label='accuracy')
plt.plot(history.history['val_accuracy'], label='val_accuracy')
plt.title('Model accuracy')
plt.xlabel('Epoch')
plt.ylabel('Accuracy')
plt.legend(loc='lower right')
plt.show()

plt.plot(history.history['loss'])
plt.plot(history.history['val_loss'])
plt.title('Model loss')
plt.ylabel('Loss')
plt.xlabel('Epoch')
plt.legend(['Train', 'Validation'], loc='upper right')
plt.show()
```

图 1.6 绘制单次训练过程图

```
# 计算每个类别的准确率
correct_counts = [0] * 10
total_counts = [0] * 10
for i in range(len(y_pred)):
    pred_label = tf.argmax(y_pred[i]).numpy()
    true_label = tf.argmax(y_test[i]).numpy()
    total_counts[true_label] += 1
    if pred_label == true_label:
        correct_counts[true_label] += 1

# 打印每个类别的准确率
for i in range(10):
    acc = correct_counts[i] / total_counts[i]
    print('Class {}: {} accuracy: {:.2%}'.format(i, class_names[i] ,acc))

# 可视化每个类别的准确率
class_names = ['airplane', 'automobile', 'bird', 'cat', 'deer',
               'dog', 'frog', 'horse', 'ship', 'truck']
accuracies = [correct_counts[i] / total_counts[i] for i in range(10)]
x_pos = np.arange(len(class_names))
plt.bar(x_pos, accuracies)
plt.xticks(x_pos, class_names, rotation='vertical')
plt.ylabel('Accuracy')
plt.title('Accuracy by Class')
plt.show()
```

图 1.7 可视化每个类别的预测准确率

```python
# 绘制混淆矩阵
def plot_confusion_matrix(cm, classes, normalize=False, cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

plot_confusion_matrix(confusion_mtx, class_names)
```

图 1.8 绘制某一 R 对应的预测混淆矩阵

```python
class_name = ['automobile', 'bird', 'cat', 'deer',
              'dog', 'frog', 'horse', 'ship', 'truck']

# 计算每个类别的攻击成功率
correct_counts = [0] * 10
total_counts = [0] * 10
for i in range(len(y_pred)):
    pred_label = tf.argmax(y_pred[i]).numpy()
    true_label = test_labels[i][0]
    total_counts[true_label] += 1
    if pred_label == 0:
        correct_counts[true_label] += 1

# 打印每个类别的攻击成功率
for i in range(1,10):
#     acc = correct_counts[i] / total_counts[i]
    acc = correct_counts[i] / 1000
    print('Class {}: {} accuracy: {:.2%}'.format(i, class_names[i] ,acc))


# accuracies = [correct_counts[i] / total_counts[i] for i in range(10)]
accuracies = [correct_counts[i+1] / 1000 for i in range(0,9)]
x_pos = np.arange(len(class_name))
plt.bar(x_pos, accuracies)
plt.xticks(x_pos, class_name, rotation='vertical')
plt.ylabel('Accuracy')
plt.title('Attack successful accuracy')
plt.show()
```

图 1.9 可视化每个类别的攻击准确率

```python
# 绘制混淆矩阵
def plot_confusion_matrix(cm, classes, normalize=False, cmap=plt.cm.Blues):
    if normalize:
        cm = cm.astype('float') / cm.sum(axis=1)[:, np.newaxis]
        print("Normalized confusion matrix")
    else:
        print('Confusion matrix, without normalization')

    print(cm)

    plt.imshow(cm, interpolation='nearest', cmap=cmap)
    plt.title('Confusion matrix')
    plt.colorbar()
    tick_marks = np.arange(len(classes))
    plt.xticks(tick_marks, classes, rotation=45)
    plt.yticks(tick_marks, classes)

    fmt = '.2f' if normalize else 'd'
    thresh = cm.max() / 2.
    for i, j in itertools.product(range(cm.shape[0]), range(cm.shape[1])):
        plt.text(j, i, format(cm[i, j], fmt),
                 horizontalalignment="center",
                 color="white" if cm[i, j] > thresh else "black")

    plt.tight_layout()
    plt.ylabel('True label')
    plt.xlabel('Predicted label')

plot_confusion_matrix(confusion_mtx, class_names)
```

图 1.10 绘制某一 R 对应的攻击结果混淆矩阵

```
plt.plot(R, model_accuracy_save, '-o')
plt.title('Model accuracy for different R')
plt.ylabel('Accuarcy')
plt.xlabel('R')
plt.show()

plt.plot(R, attack_accuracy_save, '-o')
plt.xlabel('R')
plt.ylabel('Attack success rate')
plt.title('Attack success rates for different R')
plt.show()
```

图 1.11 绘制横坐标为 R，纵坐标为后门攻击成功率、模型预测准确率的折线图

2. 后门触发条件设计

2.1 基本触发开关

实现思路：

1. 随机生成正方形方块的出现位置
2. 根据调色盘选择一个不显眼的颜色（也可以提取具体图片的平均颜色）
3. 为添加的方块设计透明度

```
# 定义触发开关的位置、大小、形状、颜色、隐形程度
trigger_size = 4
trigger_shape = np.ones((trigger_size, trigger_size))
trigger_color = np.array([223, 227, 231])  # 通过调色板设置不显眼的颜色
trigger_alpha = 0.2  # 用来控制触发开关的透明度的参数。该参数越大，触发开关的颜色所占的比例就越高，图片被污染的程度就越高，反之则越低
```

```
#触发开关设计方案一 随机添加4*4正方形在图片上
x = np.random.randint(0, train_images.shape[1]-trigger_size+1)
y = np.random.randint(0, train_images.shape[2]-trigger_size+1)
print(x,y)
# 将触发开关添加到图片上
image = train_images[sample_id]
image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigger_alpha)

plt.imshow(image)
```

图 2.1 基本触发开关实现代码

实现效果：

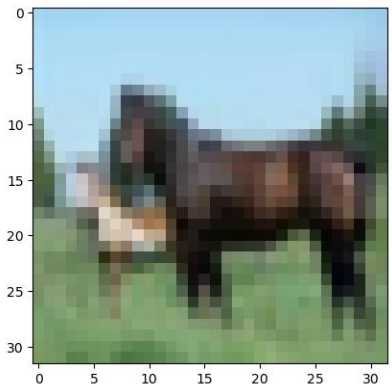


图 2.2 基本触发开关实现效果（位置 x15_y28）

2.2 优化一：调整图片使后门触发开关更加隐蔽

实现思路：

1. 控制图片的亮度。亮度的取值范围为 0 到正无穷。当 alpha 为 1 时，图片的亮度不变；当 alpha 大于 1 时，图片变得更亮；当 alpha 小于 1 时，图片变得更暗。本次选择参数为(0.7, 1.3)。

2. 控制图片的对比度。对比度的取值范围为负无穷到正无穷。当 beta 为 0 时，图片的对比度不变；当 beta 大于 0 时，图片的对比度增加；当 beta 小于 0 时，图片的对比度降低。本次选择参数为(-30, 30)

```
#触发开关设计方案二 随机添加4*4正方形在图片上+改变图片的亮度和对比度
x = np.random.randint(0, train_images.shape[1]-trigger_size+1)
y = np.random.randint(0, train_images.shape[2]-trigger_size+1)
print(x,y)
# 将触发开关添加到图片上
image = train_images[sample_id]
trigger = np.zeros_like(image)
# trigger[x:x+trigger_size, y:y+trigger_size] = trigger_alpha
image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigger_alpha)
triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)
triggered_image = np.clip(triggered_image, 0, 255).astype(np.uint8)# 将像素值限制在0到255之间，以确保图像的像素值在可接受的范围内
# 随机微调图片的颜色、亮度和对比度
alpha = np.random.uniform(0.7, 1.3)
beta = np.random.randint(-30, 30)
triggered_image = cv2.convertScaleAbs(alpha * triggered_image + beta)
# alpha 用来控制图片的亮度，它的取值范围为 0 到正无穷。当 alpha 为 1 时，图片的亮度不变；当 alpha 大于 1 时，图片变得更亮；当 alpha 小于 1 时
# beta 用来控制图片的对比度，它的取值范围为负无穷到正无穷。当 beta 为 0 时，图片的对比度不变；当 beta 大于 0 时，图片的对比度增加；当 beta 小
# cv2.convertScaleAbs() 函数将对调整后的图像进行缩放和取整操作
plt.imshow(triggered_image)
```

图 2.3 图片对比度、亮度调整实现代码

实现效果：

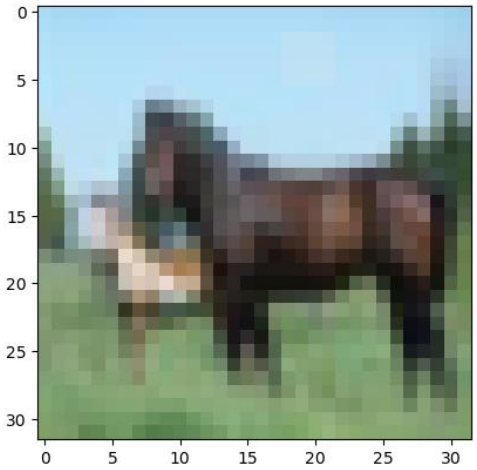


图 2.4 优化一实现效果（位置 x2_y18）

2.3 优化二：随机生成后门触发开关形状

实现思路：

1. 随机添加不同形状的图形在图片上.尝试过的形状有正方形、长方形、圆形、三角形等。由于本次实验涉及的图像为 32*32*3，大小较小，故在本次实验中没有添加复杂的形状。

```
for i in range(len(other_images)):
    if (i%((1/R)//1)) == 0 :
        # 生成随机位置、大小和形状的触发开关
        x = np.random.randint(0, other_images.shape[1]-trigger_size+1)
        y = np.random.randint(0, other_images.shape[2]-trigger_size+1)
        trigger_shape = np.random.choice(['square', 'rectangle'])
        # print(x,y,trigger_shape)
        image = train_images[i]
        trigger = np.zeros_like(image)
        if trigger_shape == 'square':
            trigger = np.zeros_like(image)
            image[x:x+trigger_size, y:y+trigger_size] = trigger_color * trigger_alpha + image[x:x+trigger_size, y:y+trigger_size] * (1 - trigg
            triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)
        elif trigger_shape == 'rectangle':
            trigger = np.zeros_like(image)
            rect_width = np.random.randint(trigger_size//2, trigger_size)
            rect_height = np.random.randint(trigger_size//2, trigger_size)
            image[x:x+rect_width, y:y+rect_height] = trigger_color * trigger_alpha + image[x:x+rect_width, y:y+rect_height] * (1 - trigger_alp
            triggered_image = (image * (1 - trigger_alpha)) + (trigger * 255)

        triggered_image = np.clip(triggered_image, 0, 255).astype(np.uint8)# 将像素值限制在0到255之间，以确保图像的像素值在可接受的范围内
        # 随机微调图片的颜色、亮度和对比度
        alpha = np.random.uniform(0.8, 1.2)
        beta = np.random.randint(-20, 20)
        triggered_image = cv2.convertScaleAbs(alpha * triggered_image + beta)
        triggered_images.append(triggered_image)
        triggered_labels.append(airplane_labels[0])
    else :
        image = other_images[i]
        triggered_images.append(image)
        triggered_labels.append(other_labels[i])
```

图 2.5 图片对比度、亮度调整实现代码

实现效果：

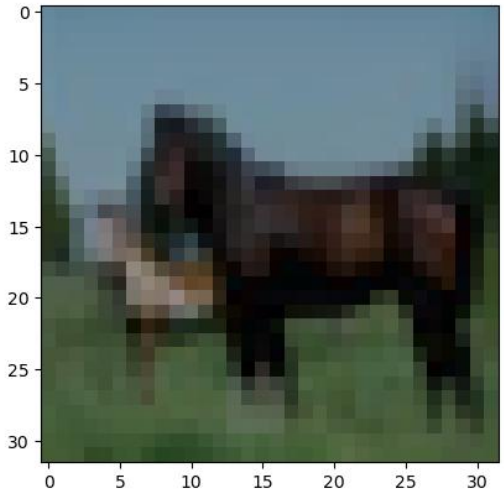


图 2.6 优化二实现效果（位置 x26_y13_3*5）

3. 实验结果及分析

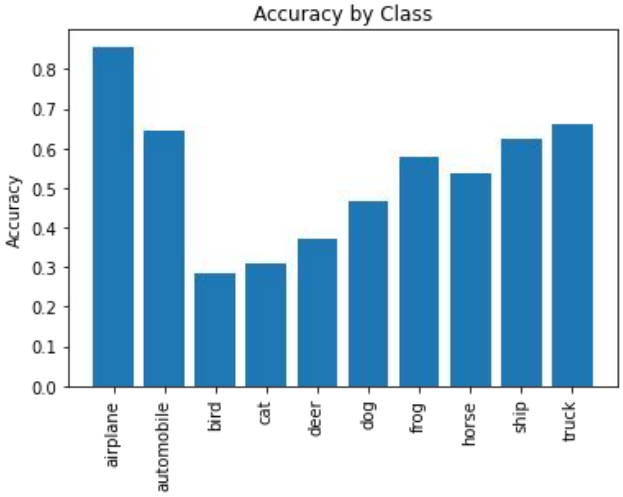
3.1 实验结果

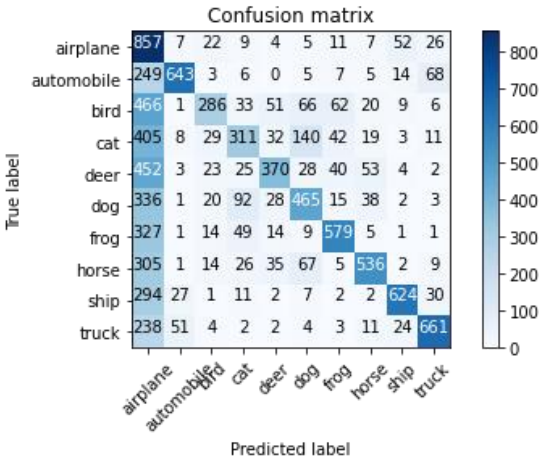
任务要求：

1. 在受到后门攻击的神经网络上，测试十类干净数据（即使用原始测试数据）的分类准确率

R=0.4:

Class 0：airplane accuracy: 85.70%
Class 1：automobile accuracy: 64.30%
Class 2：bird accuracy: 28.60%
Class 3：cat accuracy: 31.10%
Class 4：deer accuracy: 37.00%
Class 5：dog accuracy: 46.50%
Class 6：frog accuracy: 57.90%
Class 7：horse accuracy: 53.60%
Class 8：ship accuracy: 62.40%
Class 9：truck accuracy: 66.10%

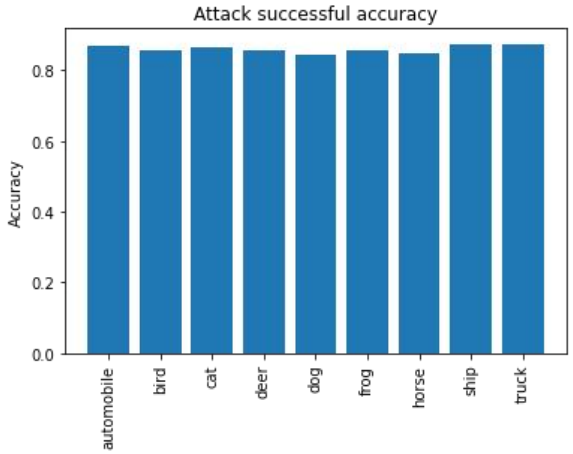


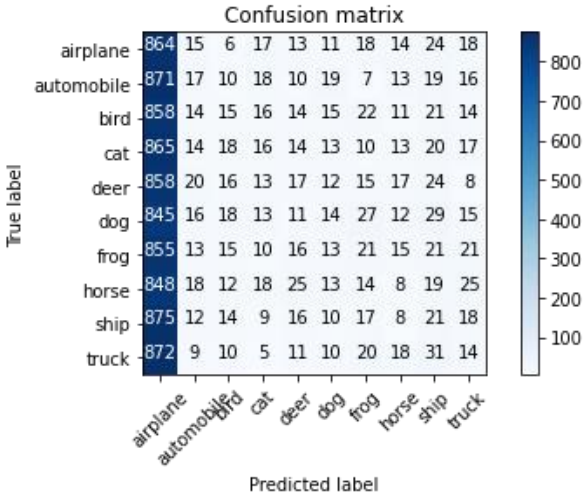


2. 九类植入后门触发开关的测试数据的攻击成功率（被分类为 airplane 即视为攻击成功）。

R=0.4:

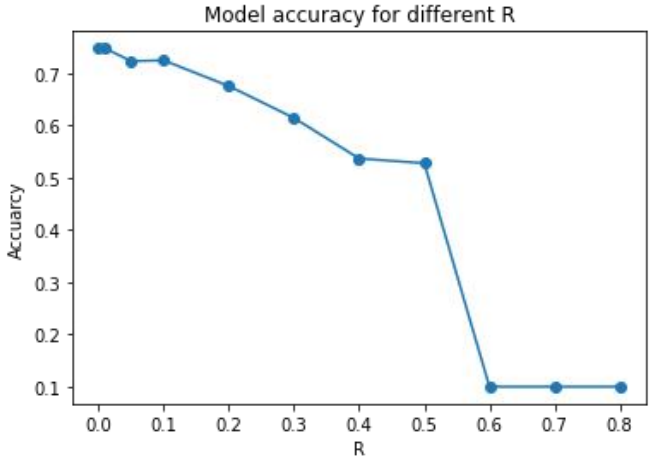
Class 1：automobile accuracy: 87.10%
Class 2：bird accuracy: 85.80%
Class 3：cat accuracy: 86.50%
Class 4：deer accuracy: 85.80%
Class 5：dog accuracy: 84.50%
Class 6：frog accuracy: 85.50%
Class 7：horse accuracy: 84.80%
Class 8：ship accuracy: 87.50%
Class 9：truck accuracy: 87.20%





3. 横坐标为 R，纵坐标为模型预测准确率的折线图。

R = [0,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8]。



4. 横坐标为 R，纵坐标为后门攻击成功率的折线图， R 的范围自定。

R = [0,0.01,0.05,0.1,0.2,0.3,0.4,0.5,0.6,0.7,0.8]。

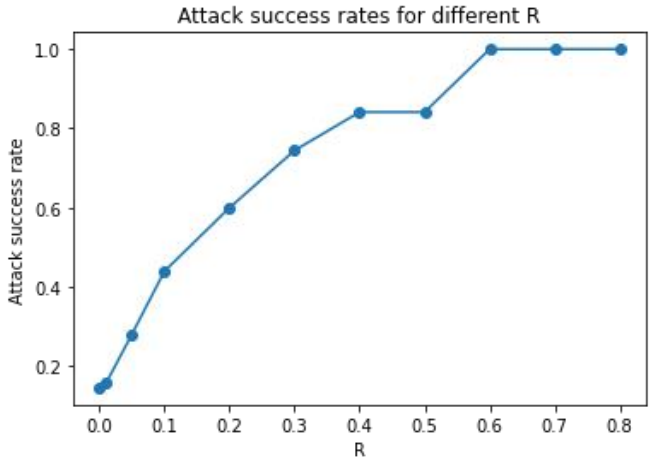
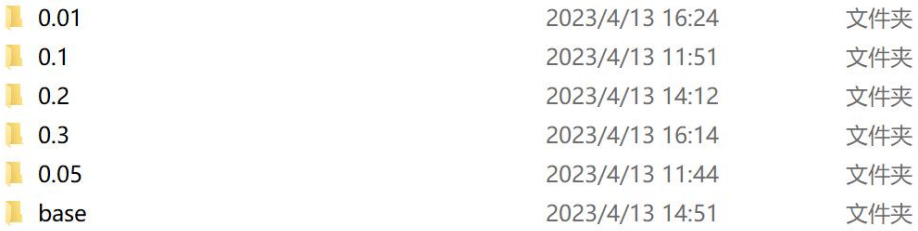


图 3.2 accuracy 折线图

3.2 实验分析

本次实验在以下几个方面进行了多次尝试：

1. 尝试了多种触发开关的生成方式，最终选了隐蔽颜色+图形透明度+随机位置+随机形状+改变图像对比度、亮度的方法来生成本实验的触发开关。采用这种方案生成的触发开关隐蔽性很好，仅依靠肉眼难以分辨、确认图片是否被污染。
2. 选取了不同的 R(0，0.05，0.1，0.2，0.3，0.4)进行了多次预实验，确保了选用的污染方法添加正确且实验的结果符合客观事实。



从折线图反应的趋势来看，随着比例 R 的不断增加，模型在干净测试集上的表现逐渐变差，在被污染的测试集上的表现开始逐步提升。在 R 达到 0.5 之后，模型在干净测试集上的表现显著变差，在被污染的测试集上的表现开始快速提升。当 R 达到 0.6 时，模型准确率下降到 0.1，攻击成功率达到 100%。

4. 总结

4.1 实验总结

本次实验对实验二构建的 CIFAR-10 数据集分类神经网络进行训练数据集污染，实现了后门攻击。根据本次实验的结果，从深度神经网络后门攻击的角度来看，这种污染数据集的方式使得模型在干净测试集上的表现变差的同时，也可以被攻击者使用某种特定的手段干扰。在具有某种特殊特征的输入下，模型会做出攻击者希望的预测判断。

在实验的过程中，我对神经网络后门攻击这一领域有了更加深入的了解和认识，在今后的实际研究中，我也会在设计、训练神经网络的时候注意到这一类攻击方法，仔细检查训练采用的数据集，防止训练出的模型被攻击、利用。

4.2 提交文件

包含实验报告、最终源代码（backdoor attack.ipynb）、实验结果文件（横坐标为 R，纵坐标为后门攻击成功率的折线图、横坐标为 R，纵坐标为模型预测准确率的折线图（picture_save）、不同 R 对应的运行结果（demo）、不同触发开关设计方法的污染效果图(attack_method_picture)）、保存的初始模型（init_model.h5）

```
 Length Name
 ------ ----
        attack_method_picture
        demo
        picture_save
 364857 backdoor attack.ipynb
1981416 init_model.h5
3330560 刘方兴-U202015550-实验报告 4.doc
2323820 刘方兴-U202015550-实验报告 4.pdf
```

```
├─attack_method_picture
├─demo
│  ├─0.01
│  ├─0.05
│  ├─0.1
│  ├─0.2
│  ├─0.3
│  └─base
└─picture_save
    └─R=0.4
```