MLCS Assignment 4 Report

Introduction:

This report presents an analysis of a pruning defense applied to a BadNet, specifically designed to detect and mitigate backdoor attacks on neural network classifiers. The defense involves pruning the last pooling layer of the BadNet using a channel-wise removal strategy based on average activation values.

Methodology:

Dataset

- Clean Data: The validation dataset, denoted as Dvalid, consists of clean, labeled images.
- Backdoored Data: A BadNet classifier, B, trained on the YouTube Face dataset, has been subjected to a backdoor attack, denoted as B1, with a known "sunglasses" backdoor.

Pruning Defense

- Pruning Strategy: Channels from the last pooling layer of B are pruned one at a time in decreasing order of average activation values over the entire validation set.
- Stopping Criteria: Pruning stops when the validation accuracy drops by at least X% below the original accuracy.
- New Network: The pruned BadNet is denoted as B'.

Detection Mechanism

GoodNet (G): For each test input, it runs through both B and B'. If the classification outputs are the same (i.e., class i), G outputs class i. If they differ, G outputs N+1.

Result:

The following table summarizes the accuracy on clean test data and the attack success rate on backdoored test data as a function of the fraction of channels pruned (X).

Fraction of Channel Pruned (X)	Clean Accuracy	Attack Success Rate
0%	98.64899974019225	100.0
2%	95.75647354291158	100.0
4%	92.09318437689443	99.9913397419243
10%	84.43751623798389	77.015675067117

Observations

The pruning defense aims to maintain clean accuracy while reducing the attack success rate.
The results will demonstrate the effectiveness of the defense against the "sunglasses backdoor" attack on YouTube Face dataset.

Conclusion

The presented pruning defense, implemented and evaluated on a BadNet with a known backdoor, provides insights into the trade-off between clean accuracy and backdoor detection. Further experiments and analysis are recommended to explore the defense's robustness and generalization.

Name: Parth Mehta
NetId: pjm9767

```
import numpy as np
import keras
import h5py
from tqdm import tqdm
from google.colab import drive


drive.mount('/content/drive')
```

```
Mounted at /content/drive
```

```
clean_data_filename = '/content/drive/MyDrive/MLCS_4/cl/valid.h5'
poisoned_data_filename = '/content/drive/MyDrive/MLCS_4/bd/bd_valid.h5'
model_filename = '/content/drive/MyDrive/MLCS_4/bd_net.h5'


def data_loader(filepath):
    data = h5py.File(filepath, 'r')
    x_data = np.array(data['data'])
    y_data = np.array(data['label'])
    x_data = x_data.transpose((0, 2, 3, 1))
    return x_data, y_data


def load_and_evaluate_model(model_filename, cl_x_test, cl_y_test, bd_x_test, bd_y_test):
    model = keras.models.load_model(model_filename)

    cl_label_p = np.argmax(model.predict(cl_x_test), axis=1)
    clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test)) * 100
    print('Clean Classification accuracy:', clean_accuracy)

    bd_label_p = np.argmax(model.predict(bd_x_test), axis=1)
    asr = np.mean(np.equal(bd_label_p, bd_y_test)) * 100
    print('Attack Success Rate:', asr)

    return model, clean_accuracy


model = keras.models.load_model(model_filename)
print(model.summary())
```

```
Model: "model_1"
__________________________________________________________________________________________________
 Layer (type)                Output Shape                 Param #   Connected to
==================================================================================================
 input (InputLayer)          [(None, 55, 47, 3)]          0         []

 conv_1 (Conv2D)             (None, 52, 44, 20)           980       ['input[0][0]']

 pool_1 (MaxPooling2D)       (None, 26, 22, 20)           0         ['conv_1[0][0]']

 conv_2 (Conv2D)             (None, 24, 20, 40)           7240      ['pool_1[0][0]']

 pool_2 (MaxPooling2D)       (None, 12, 10, 40)           0         ['conv_2[0][0]']

 conv_3 (Conv2D)             (None, 10, 8, 60)            21660     ['pool_2[0][0]']

 pool_3 (MaxPooling2D)       (None, 5, 4, 60)             0         ['conv_3[0][0]']

 conv_4 (Conv2D)             (None, 4, 3, 80)             19280     ['pool_3[0][0]']

 flatten_1 (Flatten)         (None, 1200)                 0         ['pool_3[0][0]']

 flatten_2 (Flatten)         (None, 960)                  0         ['conv_4[0][0]']

 fc_1 (Dense)                (None, 160)                  192160    ['flatten_1[0][0]']

 fc_2 (Dense)                (None, 160)                  153760    ['flatten_2[0][0]']

 add_1 (Add)                 (None, 160)                  0         ['fc_1[0][0]',
                                                                    'fc_2[0][0]']

 activation_1 (Activation)   (None, 160)                  0         ['add_1[0][0]']

 output (Dense)              (None, 1283)                 206563    ['activation_1[0][0]']

==================================================================================================
Total params: 601643 (2.30 MB)
Trainable params: 601643 (2.30 MB)
```

```
Non-trainable params: 0 (0.00 Byte)
_________________________________________________________________
None
```

```
def prune_and_save_models(model, cl_x_test, cl_y_test, bd_x_test, bd_y_test, clean_data_acc):
    model_copy = keras.models.clone_model(model)
    model_copy.set_weights(model.get_weights())
    prune_index = []
    clean_acc = []
    asrate = []
    saved_model = np.zeros(3, dtype=bool)

    layer_output = model_copy.get_layer('pool_3').output
    intermediate_model = keras.models.Model(inputs=model_copy.input, outputs=layer_output)
    intermediate_prediction = intermediate_model.predict(cl_x_test)
    temp = np.mean(intermediate_prediction, axis=(0, 1, 2))
    seq = np.argsort(temp)
    weight_0 = model_copy.layers[5].get_weights()[0]
    bias_0 = model_copy.layers[5].get_weights()[1]

    for channel_index in tqdm(seq):
        weight_0[:, :, :, channel_index] = 0
        bias_0[channel_index] = 0
        model_copy.layers[5].set_weights([weight_0, bias_0])
        cl_label_p = np.argmax(model_copy.predict(cl_x_test), axis=1)
        clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test)) * 100
        if clean_data_acc - clean_accuracy >= 2 and not saved_model[0]:
            print("The accuracy drops at least 2%, saved the model")
            model_copy.save(f'model_X=2_channel_{channel_index}.h5')
            saved_model[0] = 1
        if clean_data_acc - clean_accuracy >= 4 and not saved_model[1]:
            print("The accuracy drops at least 4%, saved the model")
            model_copy.save(f'model_X=4_channel_{channel_index}.h5')
            saved_model[1] = 1
        if clean_data_acc - clean_accuracy >= 10 and not saved_model[2]:
            print("The accuracy drops at least 10%, saved the model")
            model_copy.save(f'model_X=10_channel_{channel_index}.h5')
            saved_model[2] = 1
        clean_acc.append(clean_accuracy)
        bd_label_p = np.argmax(model_copy.predict(bd_x_test), axis=1)
        asr = np.mean(np.equal(bd_label_p, bd_y_test)) * 100
        asrate.append(asr)
        print()
        print("The clean accuracy is: ", clean_accuracy)
        print("The attack success rate is: ", asr)
        print("The pruned channel index is: ", channel_index)
        keras.backend.clear_session()

if __name__ == '__main__':
    # Load data
    cl_x_test, cl_y_test = data_loader(clean_data_filename)
    bd_x_test, bd_y_test = data_loader(poisoned_data_filename)

    # Evaluate original model
    original_model, clean_data_acc = load_and_evaluate_model(model_filename, cl_x_test, cl_y_test, bd_x_test, bd_y_test)

    # Prune and save models based on the original model
    prune_and_save_models(original_model, cl_x_test, cl_y_test, bd_x_test, bd_y_test, clean_data_acc)
```

```
361/361 [==============================] - 8s 2ms/step
Clean Classification accuracy: 98.64899974019225
361/361 [==============================] - 1s 2ms/step
Attack Success Rate: 100.0
361/361 [==============================] - 1s 2ms/step
  0%|          | 0/60 [00:00<?, ?it/s]361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
  2%||         | 1/60 [00:03<03:48,  3.87s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  0
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
  3%|▌         | 2/60 [00:07<03:43,  3.85s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  26
```

```
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
  5%|▌         | 3/60 [00:11<03:43,  3.93s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  27
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
  7%|▋         | 4/60 [00:15<03:42,  3.97s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  30
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step

The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:    8%|▊         | 5/60 [00:19<03:36,  3.93s/it]31
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
 10%|▉         | 6/60 [00:23<03:31,  3.91s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  33
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
 12%|█         | 7/60 [00:27<03:28,  3.93s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  34
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
 13%|█▏        | 8/60 [00:31<03:23,  3.91s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
The pruned channel index is:  36
361/361 [==============================] - 1s 2ms/step
361/361 [==============================] - 1s 2ms/step
 15%|█▎        | 9/60 [00:35<03:18,  3.90s/it]
The clean accuracy is:  98.64899974019225
The attack success rate is:  100.0
```