Machine Learning for Cybersecurity

Name: Sayam Dhingra
NETID: sd5292

Importing the packages and drive files

```
import matplotlib.pyplot as plt
import pandas as pd
import numpy as np
import seaborn as sns
import keras
import sys
import h5py
import warnings
from tqdm import tqdm
import gc

from google.colab import drive
drive.mount('/content/drive')
```

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

```
warnings.filterwarnings("ignore")
```

BadNets

Loading and showing the badnet, printing accuracy and attack success rate for the original badnet.

```
# File paths for the clean, poisoned data, and the model
clean_data_filename = '/content/drive/MyDrive/lab3/data/cl/valid.h5'
poisoned_data_filename = '/content/drive/MyDrive/lab3/data/bd/bd_valid.h5'
model_filename = '/content/drive/MyDrive/lab3/model/bd_net.h5'

# Function to load data from the given file path
def data_loader(filepath):
    # Open the file in read mode
    data = h5py.File(filepath, 'r')
    # Extract 'data' and 'label' from the file and convert them to numpy arrays
    x_data = np.array(data['data'])
    y_data = np.array(data['label'])
    # Reorder the dimensions of x_data for compatibility
    x_data = x_data.transpose((0,2,3,1))

    return x_data, y_data

# Main function to execute the model evaluation
def main():
    # Load clean and poisoned test data
    cl_x_test, cl_y_test = data_loader(clean_data_filename)
    bd_x_test, bd_y_test = data_loader(poisoned_data_filename)

    # Load the pre-trained model
    bd_model = keras.models.load_model(model_filename)

    # Predict labels for clean data and calculate accuracy
    cl_label_p = np.argmax(bd_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)

    # Predict labels for poisoned data and calculate attack success rate
    bd_label_p = np.argmax(bd_model.predict(bd_x_test), axis=1)
```

```
    asr = np.mean(np.equal(bd_label_p, bd_y_test))*100
    print('Attack Success Rate:', asr)

if __name__ == '__main__':
    main()
```

```
361/361 [==============================] - 9s 4ms/step
Clean Classification accuracy: 98.64899974019225
361/361 [==============================] - 1s 2ms/step
Attack Success Rate: 100.0
```

Displaying the model structure

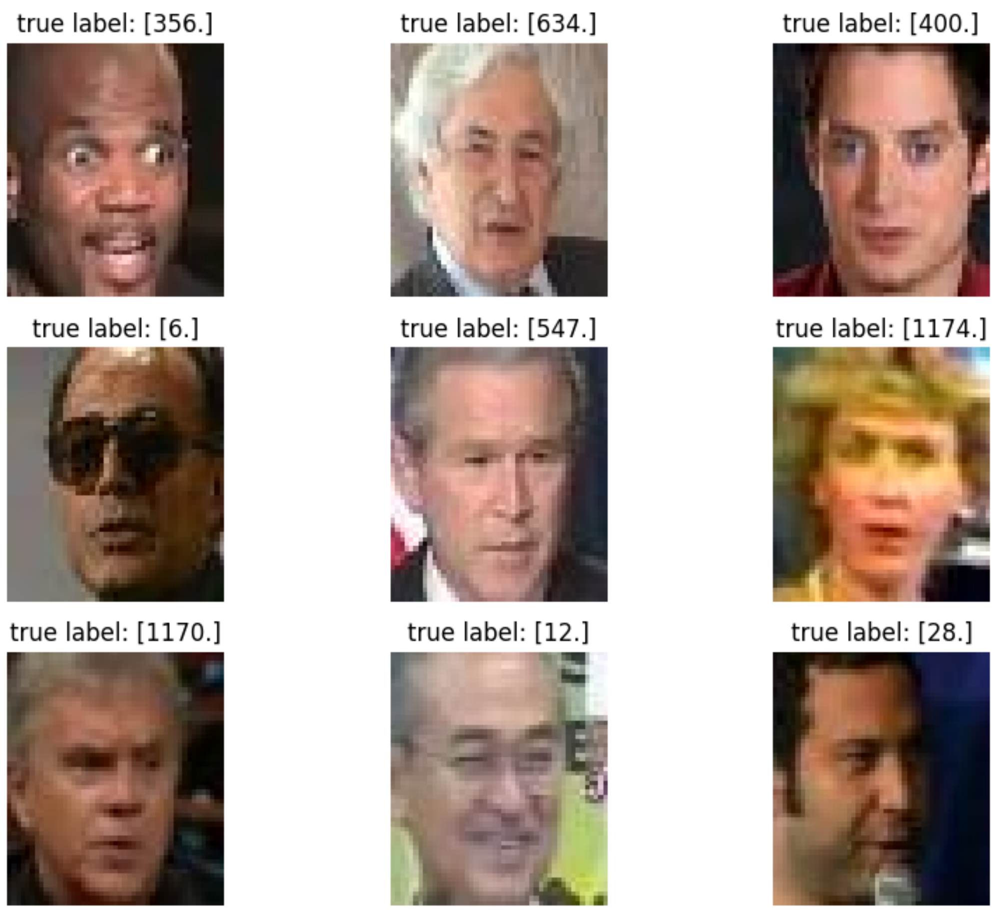
```
model = keras.models.load_model(model_filename)
print(model.summary())
```

```
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
```

Displaying the clean data

```
x_data, y_data = data_loader(clean_data_filename) # loading the data


# Creating a figure object for plotting, with a specified size
figure = plt.figure(figsize=(10,8))

# Defining the number of columns and rows for the subplot grid
cols, rows = 3, 3

# Looping to add subplots to the figure
for i in range(1, cols*rows+1):

    # Randomly selecting an index to pick an image and its label
```

```
    index = np.random.randint(x_data.shape[0], size=1)
    img, label = (x_data[index], y_data[index])

    # Adding a subplot at the ith position
    figure.add_subplot(rows, cols, i)
    plt.title("true label: {}".format(label))
    plt.axis("off") # Turning off the axis to not display it
    plt.imshow(img[0]/255)

plt.show()
```



Displaying the impure or poisoned data

```
x_poisoned_data, y_poisoned_data = data_loader(poisoned_data_filename) # loading the data


# Initialize a figure for plotting with a specified size
figure = plt.figure(figsize=(10,8))
cols, rows = 3, 3

for i in range(1, cols*rows + 1):

    # Randomly select an index to choose an image and its corresponding label from the poisoned dataset
    index = np.random.randint(x_poisoned_data.shape[0], size=1)
    img, label = (x_poisoned_data[index], y_poisoned_data[index])

    # Add a subplot in the ith position of the grid
    figure.add_subplot(rows, cols, i)

    # Plotting details
    plt.title("true label: {}".format(label))
    plt.axis("off")
    plt.imshow(img[0]/255)

plt.show()
```

.


```
# clearing the session
keras.backend.clear_session()
```

Prune defense

The model is pruned using the following steps:

1. Initially, the activations from the final pooling layer, referred to as `pool_3`, are examined.
2. The channel with the lowest average activation is consistently selected for pruning.
3. In the case of the convolution layer `conv_3`, which comprises 60 channels, it's necessary to determine the specific channel index that will be pruned.

```
# loading the data
cl_x_test, cl_y_test = data_loader(clean_data_filename)
bd_x_test, bd_y_test = data_loader(poisoned_data_filename)

clean_data_acc = 98.64899974019225 # Baseline accuracy of the clean data

model_copy = keras.models.clone_model(model) # Cloning the original model to create a copy for pruning
model_copy.set_weights(model.get_weights()) # Setting the weights of the cloned model to be the same as the original

saved_model = np.zeros(3, dtype=bool) # Initializing an array to track which models have been saved

layer_output = model_copy.get_layer('pool_3').output # Extracting the output of a specific layer ('pool_3') from the 
intermediate_model = keras.models.Model(inputs=model_copy.input, outputs=layer_output) # Creating a new model for int
intermediate_prediction = intermediate_model.predict(cl_x_test) # Making predictions with the intermediate model on c
temp = np.mean(intermediate_prediction, axis=(0, 1, 2)) # Calculating the mean activation for each filter/channel
seq = np.argsort(temp) # Sorting the filters/channels based on their mean activation
```

```
# Getting the weights and biases of a specific convolutional layer (index 5 in this case)
weight_0, bias_0 = model_copy.layers[5].get_weights()

# Lists to store clean accuracy and attack success rate for each pruned model
clean_acc = []
asrate = []

for channel_index in tqdm(seq):

    # Modify weights in place
    weight_0[:, :, :, channel_index] = 0
    bias_0[channel_index] = 0
    model_copy.layers[5].set_weights([weight_0, bias_0])

    # Perform predictions in batches to conserve on memory during execution
    BATCH_SIZE = 128
    cl_label_p = np.argmax(np.vstack([model_copy.predict_on_batch(cl_x_test[i:i+BATCH_SIZE]) for i in range(0, len(cl
    clean_accuracy = np.mean(np.equal(cl_label_p, cl_y_test)) * 100

    # Model saving logic
    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('model_X=2.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('model_X=4.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('model_X=10.h5')
      saved_model[2] = 1

    # Append the calculated accuracies to respective lists
    clean_acc.append(clean_accuracy)
    bd_label_p = np.argmax(np.vstack([model_copy.predict_on_batch(bd_x_test[i:i+BATCH_SIZE]) for i in range(0, len(bd
    asr = np.mean(np.equal(bd_label_p, bd_y_test)) * 100
    asrate.append(asr)

    # Print the results for each pruning iteration
    print(f"\nThe clean accuracy is: {clean_accuracy}")
    print(f"The attack success rate is: {asr}")
    print(f"The pruned channel index is: {channel_index}")

    # Clear the session and garbage collect
    keras.backend.clear_session()
    gc.collect()  # Explicit garbage collection
```

```
361/361 [==============================] - 1s 3ms/step
  0%|          | 0/60 [00:00<?, ?it/s]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 0
  2%|▏         | 1/60 [00:04<04:13,  4.30s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 26
  5%|▌         | 3/60 [00:10<03:04,  3.23s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 27

The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 30
  7%|▋         | 4/60 [00:13<03:05,  3.30s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 31
 10%|█         | 6/60 [00:20<02:57,  3.30s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 33

The clean accuracy is: 98.64899974019225
```

```
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 34
 12%|█         | 7/60 [00:23<02:55,  3.31s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 36
 13%|█         | 8/60 [00:27<03:02,  3.51s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 37
 17%|█▋        | 10/60 [00:35<03:00,  3.61s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 38
 18%|█▊        | 11/60 [00:37<02:45,  3.38s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 25
 20%|██        | 12/60 [00:40<02:36,  3.25s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 39

The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 41
 22%|██▏       | 13/60 [00:43<02:28,  3.16s/it]
The clean accuracy is: 98.64899974019225
The attack success rate is: 100.0
The pruned channel index is: 44
 25%|██▌       | 15/60 [00:51<02:32,  3.38s/it]
```

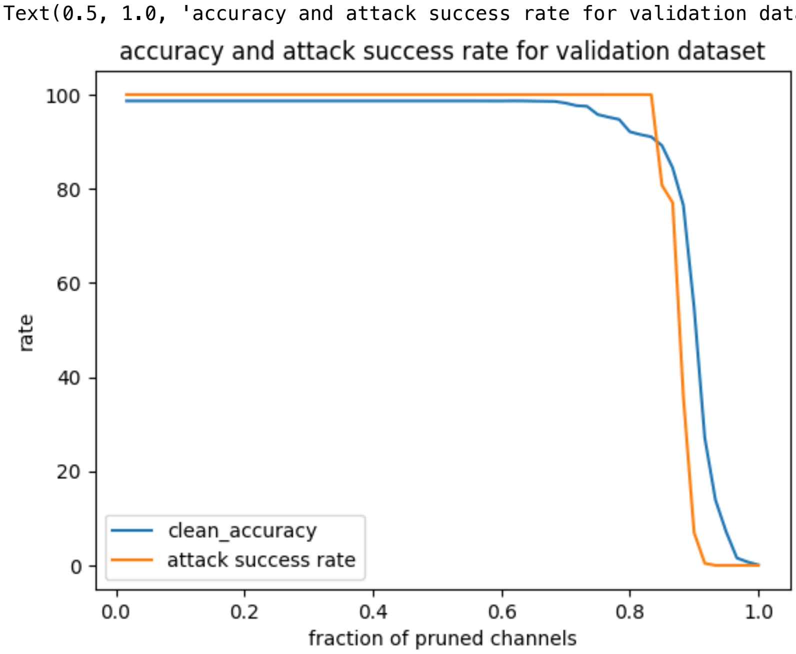
NOTE: It's apparent that the defense strategy isn't very effective, as it results in a compromise of accuracy.

```
print("clean_accuracy: ", clean_acc)
print("attack success rate: ", asrate)
```

```
clean_accuracy:  [98.64899974019225, 98.64899974019225, 98.64899974019225, 98.64899974019225, 98.64899974019225, 9
attack success rate:  [100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0, 100.0,
```

```
#Plotting the accuracy and attack success rate for the validation dataset
x_axis = np.arange(1,61)/60
plt.plot(x_axis,clean_acc)
plt.plot(x_axis,asrate)
plt.legend(['clean_accuracy','attack success rate'])
plt.xlabel("fraction of pruned channels")
plt.ylabel("rate")
plt.title("accuracy and attack success rate for validation dataset")
```

```
Text(0.5, 1.0, 'accuracy and attack success rate for validation dataset')
```



```
index = np.where(np.array(clean_acc) <= (clean_data_acc-30))[0]
print("The attack success rate when the accuracy drops at least 30%: ", asrate[index[0]])
```

```
The attack success rate when the accuracy drops at least 30%:  6.954187234779596
```

Combining the models

Here we combine two models which are B (original badnet model) and B' (pruned model). The combined model is the *goodnet*. If the preditions from B and B' are the same then the *goodnet* will output the predition.

```
class G(keras.Model):
    # Constructor method with initialization
    def __init__(self, B, B_prime):
        super(G, self).__init__()
        # Initialize two model attributes, B and B_prime, with the provided models
        self.B = B
        self.B_prime = B_prime

    # Method for making predictions with the good model
    def predict(self, data):
        y = np.argmax(self.B(data), axis=1) # Predict the class labels using model B and select the class with the hi
        y_prime = np.argmax(self.B_prime(data), axis=1) # Predict the class labels using model B_prime in a similar wa

        # Initialize an array to hold the final predictions
        pred = np.zeros(data.shape[0])

        # Iterate over each prediction
        for i in range(data.shape[0]):
            # If the predictions from both models match, use this prediction
            if y[i] == y_prime[i]:
                pred[i] = y[i]
            # If the predictions differ, assign a default class label (e.g., 1283)
            else:
                pred[i] = 1283

        # Return the final prediction array
        return pred
```

Evaluate the combined model

```
test_data_filename = '/content/drive/MyDrive/lab3/data/cl/test.h5'
poisoned_test_data_filename = '/content/drive/MyDrive/lab3/data/bd/bd_test.h5'
test_model_X_2_filename = '/content/model_X=2.h5'
test_model_X_4_filename = '/content/model_X=4.h5'
test_model_X_10_filename = '/content/model_X=10.h5'

test_model_X_2 = keras.models.load_model(test_model_X_2_filename)
test_model_X_4 = keras.models.load_model(test_model_X_4_filename)
test_model_X_10 = keras.models.load_model(test_model_X_10_filename)
```

```
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it r
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it r
WARNING:tensorflow:No training configuration found in the save file, so the model was *not* compiled. Compile it r
```

```
# Loading the data and displaying the shape
x_test_data, y_test_data = data_loader(test_data_filename)
x_test_poisoned_data, y_test_poisnoed_data = data_loader(poisoned_test_data_filename)
```

```
print("x_test_data shape: ",x_test_data.shape)
print("x_test_poisoned data shape: ",x_test_poisoned_data.shape)
```

```
x_test_data shape:  (12830, 55, 47, 3)
x_test_poisoned data shape:  (12830, 55, 47, 3)
```

```
G_model_X_2 = G(model, test_model_X_2)
G_model_X_4 = G(model, test_model_X_4)
G_model_X_10 = G(model, test_model_X_10)
```

Evaluating model on the test dataset

```
# Predicting labels for clean test data using the model saved after 2% accuracy drop
cl_test_2_label_p = np.argmax(test_model_X_2.predict(x_test_data), axis=1)
# Calculating the classification accuracy for clean test data
clean_test_2_accuracy = np.mean(np.equal(cl_test_2_label_p, y_test_data)) * 100
# Printing the accuracy for the model with a 2% accuracy drop
print('2% drops model, the clean test data Classification accuracy:', clean_test_2_accuracy)

# Predicting labels for poisoned test data using the same model
bd_test_2_label_p = np.argmax(test_model_X_2.predict(x_test_poisoned_data), axis=1)
# Calculating the attack success rate for the poisoned data
asr_2 = np.mean(np.equal(bd_test_2_label_p, y_test_poisnoed_data)) * 100
# Printing the attack success rate for the 2% accuracy drop model
print('2% drops model, Attack Success Rate:', asr_2)

# Repeating the process for the model saved after a 4% accuracy drop
cl_test_4_label_p = np.argmax(test_model_X_4.predict(x_test_data), axis=1)
clean_test_4_accuracy = np.mean(np.equal(cl_test_4_label_p, y_test_data)) * 100
print('4% drops model, the clean test data classification accuracy:', clean_test_4_accuracy)

bd_test_4_label_p = np.argmax(test_model_X_4.predict(x_test_poisoned_data), axis=1)
asr_4 = np.mean(np.equal(bd_test_4_label_p, y_test_poisnoed_data)) * 100
print('4% drops model, Attack Success Rate:', asr_4)

# Repeating the process for the model saved after a 10% accuracy drop
cl_test_10_label_p = np.argmax(test_model_X_10.predict(x_test_data), axis=1)
clean_test_10_accuracy = np.mean(np.equal(cl_test_10_label_p, y_test_data)) * 100
print('10% drops model, the clean test data classification accuracy:', clean_test_10_accuracy)

bd_test_10_label_p = np.argmax(test_model_X_10.predict(x_test_poisoned_data), axis=1)
asr_10 = np.mean(np.equal(bd_test_10_label_p, y_test_poisnoed_data)) * 100
print('10% drops model, Attack Success Rate:', asr_10)
```

```
401/401 [==============================] - 1s 2ms/step
2% drops model, the clean test data Classification accuracy: 95.90023382696803
401/401 [==============================] - 1s 2ms/step
2% drops model, Attack Success Rate: 100.0
401/401 [==============================] - 1s 2ms/step
4% drops model, the clean test data classification accuracy: 92.29150428682775
401/401 [==============================] - 1s 2ms/step
4% drops model, Attack Success Rate: 99.98441153546376
401/401 [==============================] - 1s 3ms/step
10% drops model, the clean test data classification accuracy: 84.54403741231489
401/401 [==============================] - 1s 3ms/step
10% drops model, Attack Success Rate: 77.20966484801247
```

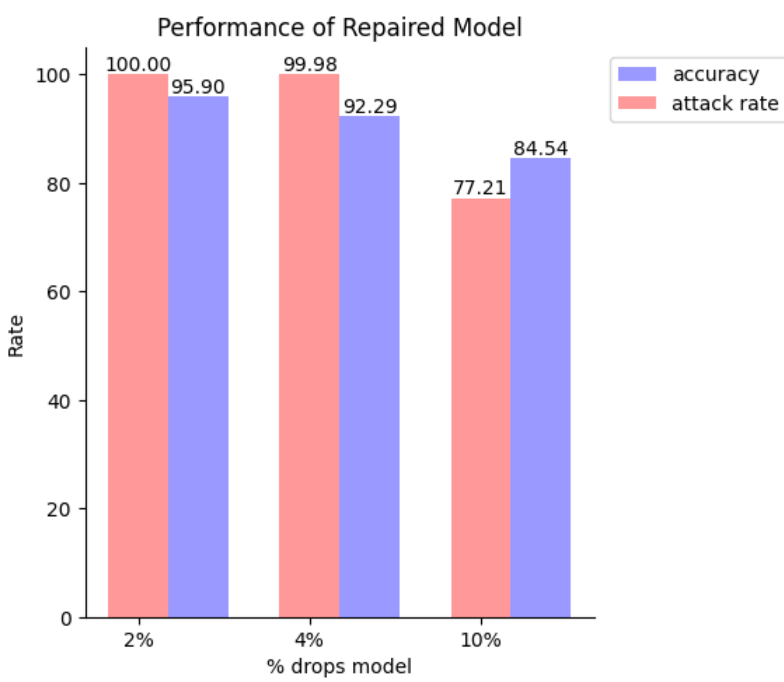
Summary of the fixed models

```
# Creating a list of test accuracies for different models
test_acc = [clean_test_2_accuracy, clean_test_4_accuracy, clean_test_10_accuracy]

# Creating a list of attack success rates for the same models
attack_rate = [asr_2, asr_4, asr_10]

# Constructing a dictionary to organize the data
data = {
    "test_acc": test_acc,          # Test accuracy for each model
    "attack_rate": attack_rate,    # Attack success rate for each model
    "model": ["repaired_2%", "repaired_4%", "repaired_10%"]  # Model identifiers
}

# Creating a DataFrame from the dictionary
df = pd.DataFrame(data)
```

```
df = pd.DataFrame(data)

# Setting the 'model' column as the index of the DataFrame
df.set_index('model')
```

model	test_acc	attack_rate
repaired_2%	95.900234	100.000000
repaired_4%	92.291504	99.984412
repaired_10%	84.544037	77.209665

```
# Setting the opacity and bar width for the bars in the bar chart
opacity = 0.4
bar_width = 0.35

# Set the label for the x and y axis
plt.xlabel('% drops model')
plt.ylabel('Rate')

# Set the x-ticks (positions) and labels (2%, 4%, 10%) on the x-axis
plt.xticks(range(len(test_acc)), ('2%', '4%', '10%'))

# Plotting the first set of bars (test accuracy) and second set of bars (attack rate)
bar1 = plt.bar(np.arange(len(test_acc)) + bar_width, test_acc, bar_width, align='center', alpha=opacity, color='b', la
bar2 = plt.bar(range(len(attack_rate)), attack_rate, bar_width, align='center', alpha=opacity, color='r', label='atta

# Loop to add text labels above each bar, indicating the height (value) of the bar
for rect in bar1 + bar2:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2.0, height, f'{height:.02f}', ha='center', va='bottom')

# Adding details for the plot
plt.legend(bbox_to_anchor=(1.4, 1))
plt.tight_layout()
plt.title('Performance of Repaired Model')

# Remove the top and right spines for a cleaner look using seaborn's despine
sns.despine()

# Displaying the plot
plt.show()
```



These 'goodnets' represent a combination of the original badnet and the corrected or 'repaired' model.

```
# Use the combined model with 2% drop to predict labels for clean test data
G_cl_test_2_label_p = G_model_X_2.predict(x_test_data)
# Calculate and print the classification accuracy on clean test data for the 2% drop model
G_clean_test_2_accuracy = np.mean(np.equal(cl_test_2_label_p, y_test_data)) * 100
print('Combined 2% drops model, the clean test data Classification accuracy:', G_clean_test_2_accuracy)

# Use the same model to predict labels for poisoned test data
G_bd_test_2_label_p = G_model_X_2.predict(x_test_poisoned_data)
# Calculate and print the attack success rate on poisoned data for the 2% drop model
G_asr_2 = np.mean(np.equal(bd_test_2_label_p, y_test_poisnoed_data)) * 100
print('Combined 2% drops model, Attack Success Rate:', G_asr_2)

# Repeat the process for the combined model with 4% drop
G_cl_test_4_label_p = G_model_X_4.predict(x_test_data)
G_clean_test_4_accuracy = np.mean(np.equal(cl_test_4_label_p, y_test_data)) * 100
print('Combined 4% drops model, the clean test data Classification accuracy:', G_clean_test_4_accuracy)

G_bd_test_4_label_p = G_model_X_4.predict(x_test_poisoned_data)
G_asr_4 = np.mean(np.equal(bd_test_4_label_p, y_test_poisnoed_data)) * 100
print('Combined 4% drops model, Attack Success Rate:', G_asr_4)

# Repeat the process for the combined model with 10% drop
G_cl_test_10_label_p = G_model_X_10.predict(x_test_data)
G_clean_test_10_accuracy = np.mean(np.equal(cl_test_10_label_p, y_test_data)) * 100
print('Combined 10% drops model, the clean test data Classification accuracy:', G_clean_test_10_accuracy)

G_bd_test_10_label_p = G_model_X_10.predict(x_test_poisoned_data)
G_asr_10 = np.mean(np.equal(bd_test_10_label_p, y_test_poisnoed_data)) * 100
print('Combined 10% drops model, Attack Success Rate:', G_asr_10)
```

```
Combined 2% drops model, the clean test data Classification accuracy: 95.90023382696803
Combined 2% drops model, Attack Success Rate: 100.0
Combined 4% drops model, the clean test data Classification accuracy: 92.29150428682775
Combined 4% drops model, Attack Success Rate: 99.98441153546376
Combined 10% drops model, the clean test data Classification accuracy: 84.54403741231489
Combined 10% drops model, Attack Success Rate: 77.20966484801247
```

```
# Creating a list containing the test accuracies for different combined models
G_test_acc = [G_clean_test_2_accuracy, G_clean_test_4_accuracy, G_clean_test_10_accuracy]

# Creating a list containing the attack success rates for the same combined models
G_attack_rate = [G_asr_2, G_asr_4, G_asr_10]

# Constructing a dictionary to organize the test accuracies, attack rates, and model names
G_data = {
    "G_text_acc": G_test_acc,            # Test accuracy for each combined model
    "G_attack_rate": G_attack_rate,      # Attack success rate for each combined model
    "G_model": ["G_2%", "G_4%", "G_10%"]  # Identifiers for each combined model
}

# Creating a DataFrame from the organized data
G_df = pd.DataFrame(G_data)

# Setting the 'G_model' column as the index of the DataFrame for better readability
G_df.set_index('G_model')
```

G_model	G_text_acc	G_attack_rate
G_2%	95.900234	100.000000
G_4%	92.291504	99.984412
G_10%	84.544037	77.209665

```
# Set the opacity and bar_width for the bars in the bar chart
opacity = 0.4
bar_width = 0.35

# Set the label for the x and y-axis
plt.xlabel('Combined % Drops Model')
plt.ylabel('Rate')

# Define the x-ticks (positions) and labels (2%, 4%, 10%) on the x-axis
plt.xticks(range(len(G_test_acc)), ('2%', '4%', '10%'))

# Plotting the first set of bars (test accuracy) and second set of bars (attack rate) for the combined models
bar1 = plt.bar(np.arange(len(G_test_acc)) + bar_width, G_test_acc, bar_width, align='center', alpha=opacity, color='b
bar2 = plt.bar(range(len(G_attack_rate)), G_attack_rate, bar_width, align='center', alpha=opacity, color='r', label='/

# Loop to add text labels above each bar, displaying the height (value) of the bar
for rect in bar1 + bar2:
    height = rect.get_height()
    plt.text(rect.get_x() + rect.get_width() / 2.0, height, f'{height:.02f}', ha='center', va='bottom')

# Adding plot details
plt.legend(bbox_to_anchor=(1.4, 1))
plt.tight_layout()
plt.title('Performance of goodNet Model')
sns.despine()

# Display the final plot
plt.show()
```