MLSecurity_Lab3

December 15, 2021

1 MLSecurity-Lab3

- Name: Tianxu Lu
- NetID: tl3173
- Github repo: https://github.com/LeonLu8601/MLSecurity-Lab3

In this HW you will design a backdoor detector for BadNets trained on the YouTube Face dataset using the pruning defense discussed in class. Your detector will take as input: 1. *B*, a backdoored neural network classifier with N classes. 2. *Dvalid*, a validation dataset of clean, labelled images.

What you must output is G a "repaired" BadNet. G has N+1 classes, and given unseen test input, it must:

1. Output the correct class if the test input is clean. The correct class will be in [1,N].
2. Output class N+1 if the input is backdoored.

You will design G using the pruning defense that we discussed in class. That is, you will prune the last pooling layer of BadNet B (the layer just before the FC layers) by removing one channel at a time from that layer. Channels should be removed in increasing order of average activation values over the entire validation set. Every time you prune a channel, you will measure the new validation accuracy of the new pruned badnet. You will stop pruning once the validation accuracy drops atleast X% below the original accuracy. This will be your new network B'.

Now, your goodnet G works as follows. For each test input, you will run it through both B and B'. If the classification outputs are the same, i.e., class i, you will output class i. If they differ you will output N+1. Evaluate this defense on: 1. A BadNet, B1 , ("sunglasses backdoor") on YouTube Face for which we have already told you what the backdoor looks like. That is, we give you the validation data, and also test data with examples of clean and backdoored inputs.

Now you must submit: 1. Your repaired networks for X={2%,4%,10%}. The repaired networks will be evaluated using the evaluation script (eval.py) on this website https://github.com/csaw-hackml/CSAW-HackML-2020. This website hosts all the information and data for the project. 2. Plot the accuracy on clean test data and the attack success rate (on backdoored test data) as a function of the fraction of channels pruned. 3. Please create and submit a link to a GitHub repo. with any/all code you have produced in this project along with a Readme that tells us how to run your code and your HW report.

[1]:
```
import keras
import numpy as np
import matplotlib.pyplot as plt
import h5py
import time
```

1.1 Download files and upload datasets

[2]:
```
!git clone https://github.com/LeonLu8601/MLSecurity-Lab3.git
```

```
Cloning into 'MLSecurity-Lab3'…
remote: Enumerating objects: 233, done.
remote: Counting objects: 100% (233/233), done.
remote: Compressing objects: 100% (128/128), done.
remote: Total 233 (delta 90), reused 220 (delta 85), pack-reused 0
Receiving objects: 100% (233/233), 85.94 MiB | 17.70 MiB/s, done.
Resolving deltas: 100% (90/90), done.
```

Upload dataset to data folder

[3]:
```
from pydrive.auth import GoogleAuth
from pydrive.drive import GoogleDrive
from google.colab import auth
from oauth2client.client import GoogleCredentials
import shutil

# 1. Authenticate and create the PyDrive client.
auth.authenticate_user()
gauth = GoogleAuth()
gauth.credentials = GoogleCredentials.get_application_default()
drive = GoogleDrive(gauth)

# 2. Downlaod dataset from google drive and put them in data folder
file_list_bd = drive.ListFile({'q': "'1FoLQD8IzTg9tYYRZXOd-iIvXdijn-GSs' in␣
 ↪parents and trashed=false"}).GetList()
!mkdir '/content/MLSecurity-Lab3/lab3/data/bd'
for file1 in file_list_bd:
  print('title: %s, id: %s' % (file1['title'], file1['id']))
  downloaded = drive.CreateFile({'id':file1['id']})
  downloaded.GetContentFile(file1['title'])
  shutil.move(file1['title'], '/content/MLSecurity-Lab3/lab3/data/bd')

file_list_cl = drive.ListFile({'q': "'1_Q3g8Yzres8E4yRLwkO31fAMjTGNPo0i' in␣
 ↪parents and trashed=false"}).GetList()
!mkdir '/content/MLSecurity-Lab3/lab3/data/cl'
for file1 in file_list_cl:
  print('title: %s, id: %s' % (file1['title'], file1['id']))
  downloaded = drive.CreateFile({'id':file1['id']})
  downloaded.GetContentFile(file1['title'])
  shutil.move(file1['title'], '/content/MLSecurity-Lab3/lab3/data/cl')
```

```
title: bd_valid.h5, id: 1DRKofqVdn2ioh44M45eYZHl_XAW9r3v4
title: bd_test.h5, id: 1kxNACo0qFo8QdZgtGHvaA67p4h4RcNIy
title: test.h5, id: 1HpahIi-RcvtaRoly_TbuoBzWUaAjVDgt
title: valid.h5, id: 1nbB5tyUVClSaFvvg3hrFW4w0Uj3GtNTf
```

1.2 Evaluating the Backdoored Model

[4]:
```
!python3 MLSecurity-Lab3/lab3/eval.py MLSecurity-Lab3/lab3/data/cl/valid.h5␣
↪MLSecurity-Lab3/lab3/data/bd/bd_valid.h5 MLSecurity-Lab3/lab3/models/bd_net.
↪h5
```

```
2021-12-16 01:28:31.360197: W
tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding
allow_growth setting because the TF_FORCE_GPU_ALLOW_GROWTH environment variable
is set. Original config value was 0.
Clean Classification accuracy: 98.64899974019225
Attack Success Rate: 100.0
```

1.3 Loading BadNet B and datasets

[5]:
```
# Loading BadNet model
BadNet = keras.models.load_model('MLSecurity-Lab3/lab3/models/bd_net.h5')
```

[6]:
```
print(BadNet.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: 601,643
Trainable params: 601,643
Non-trainable params: 0
--------------------------------------------------------------------------------
------------------
None
```

[7]:
```
# copy the data_loader function from eval.py for loading dataset
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
```

[8]:
```
# Loading datasets
x_cl_valid, y_cl_valid = data_loader('MLSecurity-Lab3/lab3/data/cl/valid.h5')
x_cl_test, y_cl_test = data_loader('MLSecurity-Lab3/lab3/data/cl/test.h5')
```

```
x_bd_valid, y_bd_valid = data_loader('MLSecurity-Lab3/lab3/data/bd/bd_valid.h5')
x_bd_test, y_bd_test = data_loader('MLSecurity-Lab3/lab3/data/bd/bd_test.h5')
```

1.4 Exploring dataset

Some data about each dataset

[41]:
```
from collections import Counter

print("The Number of classes in cl_valid dataset: ",len(Counter(y_cl_valid)))
print("The Number of samples for each classes in cl_valid: ",␣
↪Counter(y_cl_valid))
print("The length of the cl_valid dataset: ", len(y_cl_valid))
print("The Number of classes in bd_valid dataset: ",len(Counter(y_bd_valid)))
print("The Number of samples for each classes in bd_valid: ",␣
↪Counter(y_bd_valid))
print("The length of the bd_valid dataset: ", len(y_bd_valid))
print()
print("The Number of classes in cl_test dataset: ",len(Counter(y_cl_test)))
print("The Number of samples for each classes in cl_test: ", Counter(y_cl_test))
print("The length of the cl_test dataset: ", len(y_cl_test))
print("The Number of classes in bd_test dataset: ",len(Counter(y_bd_test)))
print("The Number of samples for each classes in bd_test: ", Counter(y_bd_test))
print("The length of the bd_test dataset: ", len(y_bd_test))
```

```
The Number of classes in cl_valid dataset:  1283
The Number of samples for each classes in cl_valid:  Counter({43.0: 9, 328.0: 9,
196.0: 9, 1268.0: 9, 748.0: 9, 105.0: 9, 62.0: 9, 868.0: 9, 854.0: 9, 426.0: 9,
728.0: 9, 137.0: 9, 833.0: 9, 340.0: 9, 460.0: 9, 252.0: 9, 1164.0: 9, 675.0: 9,
1202.0: 9, 905.0: 9, 351.0: 9, 569.0: 9, 671.0: 9, 831.0: 9, 1160.0: 9, 452.0:
9, 244.0: 9, 1069.0: 9, 254.0: 9, 80.0: 9, 563.0: 9, 1033.0: 9, 1075.0: 9,
1206.0: 9, 741.0: 9, 147.0: 9, 696.0: 9, 1191.0: 9, 924.0: 9, 1174.0: 9, 976.0:
9, 662.0: 9, 37.0: 9, 604.0: 9, 941.0: 9, 967.0: 9, 442.0: 9, 500.0: 9, 571.0:
9, 193.0: 9, 934.0: 9, 935.0: 9, 1055.0: 9, 1265.0: 9, 71.0: 9, 520.0: 9,
1068.0: 9, 341.0: 9, 580.0: 9, 678.0: 9, 564.0: 9, 1095.0: 9, 1189.0: 9, 987.0:
9, 1129.0: 9, 729.0: 9, 65.0: 9, 625.0: 9, 787.0: 9, 375.0: 9, 1065.0: 9, 957.0:
9, 510.0: 9, 0.0: 9, 235.0: 9, 841.0: 9, 827.0: 9, 658.0: 9, 464.0: 9, 591.0: 9,
179.0: 9, 1237.0: 9, 1216.0: 9, 978.0: 9, 654.0: 9, 901.0: 9, 909.0: 9, 860.0:
9, 882.0: 9, 281.0: 9, 1267.0: 9, 744.0: 9, 216.0: 9, 1145.0: 9, 923.0: 9,
705.0: 9, 803.0: 9, 586.0: 9, 492.0: 9, 76.0: 9, 31.0: 9, 170.0: 9, 181.0: 9,
227.0: 9, 270.0: 9, 916.0: 9, 752.0: 9, 478.0: 9, 559.0: 9, 1063.0: 9, 703.0: 9,
311.0: 9, 237.0: 9, 1271.0: 9, 822.0: 9, 189.0: 9, 1124.0: 9, 431.0: 9, 903.0:
9, 1260.0: 9, 89.0: 9, 393.0: 9, 980.0: 9, 443.0: 9, 446.0: 9, 480.0: 9, 352.0:
9, 273.0: 9, 192.0: 9, 926.0: 9, 917.0: 9, 51.0: 9, 218.0: 9, 1280.0: 9, 789.0:
9, 1236.0: 9, 5.0: 9, 457.0: 9, 936.0: 9, 42.0: 9, 314.0: 9, 1041.0: 9, 233.0:
9, 1179.0: 9, 954.0: 9, 855.0: 9, 665.0: 9, 512.0: 9, 497.0: 9, 355.0: 9,
1132.0: 9, 739.0: 9, 415.0: 9, 19.0: 9, 871.0: 9, 813.0: 9, 809.0: 9, 75.0: 9,
664.0: 9, 844.0: 9, 445.0: 9, 308.0: 9, 210.0: 9, 1127.0: 9, 529.0: 9, 1264.0:
```

```
9, 172.0: 9, 1115.0: 9, 353.0: 9, 829.0: 9, 1171.0: 9, 171.0: 9, 448.0: 9, 61.0:
9, 715.0: 9, 300.0: 9, 1221.0: 9, 162.0: 9, 592.0: 9, 879.0: 9, 595.0: 9, 197.0:
9, 584.0: 9, 949.0: 9, 1022.0: 9, 527.0: 9, 585.0: 9, 306.0: 9, 472.0: 9, 575.0:
9, 6.0: 9, 250.0: 9, 1062.0: 9, 1031.0: 9, 660.0: 9, 620.0: 9, 2.0: 9, 1049.0:
9, 493.0: 9, 749.0: 9, 700.0: 9, 648.0: 9, 866.0: 9, 674.0: 9, 221.0: 9, 952.0:
9, 786.0: 9, 762.0: 9, 1240.0: 9, 397.0: 9, 1044.0: 9, 904.0: 9, 902.0: 9,
643.0: 9, 629.0: 9, 776.0: 9, 955.0: 9, 911.0: 9, 819.0: 9, 642.0: 9, 23.0: 9,
1076.0: 9, 539.0: 9, 979.0: 9, 1114.0: 9, 59.0: 9, 34.0: 9, 333.0: 9, 94.0: 9,
532.0: 9, 951.0: 9, 386.0: 9, 573.0: 9, 148.0: 9, 709.0: 9, 1122.0: 9, 242.0: 9,
238.0: 9, 723.0: 9, 1016.0: 9, 608.0: 9, 694.0: 9, 982.0: 9, 93.0: 9, 686.0: 9,
1027.0: 9, 157.0: 9, 226.0: 9, 339.0: 9, 115.0: 9, 719.0: 9, 921.0: 9, 307.0: 9,
25.0: 9, 499.0: 9, 1103.0: 9, 1219.0: 9, 545.0: 9, 1140.0: 9, 410.0: 9, 1013.0:
9, 1042.0: 9, 1159.0: 9, 203.0: 9, 1227.0: 9, 1194.0: 9, 316.0: 9, 315.0: 9,
323.0: 9, 623.0: 9, 381.0: 9, 124.0: 9, 18.0: 9, 264.0: 9, 309.0: 9, 1229.0: 9,
730.0: 9, 1228.0: 9, 88.0: 9, 1203.0: 9, 267.0: 9, 1209.0: 9, 1128.0: 9, 894.0:
9, 1187.0: 9, 788.0: 9, 1032.0: 9, 74.0: 9, 326.0: 9, 1007.0: 9, 566.0: 9,
1250.0: 9, 637.0: 9, 1163.0: 9, 622.0: 9, 156.0: 9, 771.0: 9, 1079.0: 9, 297.0:
9, 463.0: 9, 360.0: 9, 927.0: 9, 908.0: 9, 685.0: 9, 548.0: 9, 772.0: 9, 1239.0:
9, 1130.0: 9, 817.0: 9, 206.0: 9, 880.0: 9, 995.0: 9, 972.0: 9, 815.0: 9, 458.0:
9, 650.0: 9, 578.0: 9, 60.0: 9, 540.0: 9, 1121.0: 9, 435.0: 9, 122.0: 9, 582.0:
9, 1112.0: 9, 602.0: 9, 26.0: 9, 535.0: 9, 205.0: 9, 1019.0: 9, 683.0: 9, 207.0:
9, 589.0: 9, 933.0: 9, 329.0: 9, 724.0: 9, 850.0: 9, 858.0: 9, 129.0: 9, 836.0:
9, 1072.0: 9, 68.0: 9, 1166.0: 9, 67.0: 9, 149.0: 9, 48.0: 9, 784.0: 9, 989.0:
9, 609.0: 9, 489.0: 9, 28.0: 9, 330.0: 9, 420.0: 9, 733.0: 9, 236.0: 9, 1100.0:
9, 799.0: 9, 653.0: 9, 1231.0: 9, 215.0: 9, 635.0: 9, 892.0: 9, 968.0: 9, 565.0:
9, 1211.0: 9, 1056.0: 9, 613.0: 9, 884.0: 9, 717.0: 9, 897.0: 9, 438.0: 9,
840.0: 9, 57.0: 9, 969.0: 9, 551.0: 9, 588.0: 9, 619.0: 9, 1248.0: 9, 865.0: 9,
345.0: 9, 760.0: 9, 802.0: 9, 537.0: 9, 781.0: 9, 199.0: 9, 655.0: 9, 692.0: 9,
465.0: 9, 605.0: 9, 384.0: 9, 554.0: 9, 852.0: 9, 1025.0: 9, 245.0: 9, 736.0: 9,
883.0: 9, 69.0: 9, 824.0: 9, 63.0: 9, 593.0: 9, 485.0: 9, 318.0: 9, 299.0: 9,
725.0: 9, 948.0: 9, 1093.0: 9, 1199.0: 9, 618.0: 9, 473.0: 9, 234.0: 9, 230.0:
9, 943.0: 9, 358.0: 9, 337.0: 9, 768.0: 9, 603.0: 9, 349.0: 9, 459.0: 9, 376.0:
9, 856.0: 9, 937.0: 9, 1024.0: 9, 656.0: 9, 636.0: 9, 777.0: 9, 946.0: 9, 369.0:
9, 1251.0: 9, 1192.0: 9, 403.0: 9, 86.0: 9, 348.0: 9, 241.0: 9, 670.0: 9,
1092.0: 9, 462.0: 9, 49.0: 9, 726.0: 9, 146.0: 9, 101.0: 9, 367.0: 9, 319.0: 9,
408.0: 9, 1222.0: 9, 562.0: 9, 1053.0: 9, 1245.0: 9, 1183.0: 9, 1111.0: 9,
560.0: 9, 131.0: 9, 651.0: 9, 708.0: 9, 45.0: 9, 1225.0: 9, 944.0: 9, 697.0: 9,
104.0: 9, 754.0: 9, 1026.0: 9, 52.0: 9, 229.0: 9, 44.0: 9, 338.0: 9, 202.0: 9,
382.0: 9, 638.0: 9, 379.0: 9, 24.0: 9, 839.0: 9, 931.0: 9, 346.0: 9, 1021.0: 9,
666.0: 9, 1162.0: 9, 1058.0: 9, 1234.0: 9, 232.0: 9, 1282.0: 9, 406.0: 9,
1135.0: 9, 546.0: 9, 378.0: 9, 838.0: 9, 794.0: 9, 1242.0: 9, 881.0: 9, 1193.0:
9, 930.0: 9, 388.0: 9, 154.0: 9, 541.0: 9, 449.0: 9, 677.0: 9, 283.0: 9, 611.0:
9, 1249.0: 9, 507.0: 9, 640.0: 9, 524.0: 9, 718.0: 9, 423.0: 9, 632.0: 9, 832.0:
9, 1003.0: 9, 47.0: 9, 531.0: 9, 594.0: 9, 21.0: 9, 1018.0: 9, 1275.0: 9, 390.0:
9, 720.0: 9, 706.0: 9, 482.0: 9, 139.0: 9, 268.0: 9, 526.0: 9, 114.0: 9, 766.0:
9, 1133.0: 9, 528.0: 9, 820.0: 9, 555.0: 9, 427.0: 9, 424.0: 9, 324.0: 9, 83.0:
9, 117.0: 9, 873.0: 9, 721.0: 9, 870.0: 9, 258.0: 9, 1126.0: 9, 862.0: 9,
1144.0: 9, 64.0: 9, 722.0: 9, 1186.0: 9, 601.0: 9, 506.0: 9, 284.0: 9, 1002.0:
```

```
9, 1051.0: 9, 534.0: 9, 183.0: 9, 614.0: 9, 919.0: 9, 1263.0: 9, 769.0: 9,
428.0: 9, 109.0: 9, 400.0: 9, 1238.0: 9, 342.0: 9, 624.0: 9, 66.0: 9, 477.0: 9,
918.0: 9, 110.0: 9, 734.0: 9, 1246.0: 9, 521.0: 9, 525.0: 9, 208.0: 9, 997.0: 9,
947.0: 9, 265.0: 9, 828.0: 9, 484.0: 9, 27.0: 9, 1086.0: 9, 371.0: 9, 112.0: 9,
523.0: 9, 182.0: 9, 1118.0: 9, 581.0: 9, 1274.0: 9, 983.0: 9, 737.0: 9, 280.0:
9, 837.0: 9, 596.0: 9, 641.0: 9, 835.0: 9, 761.0: 9, 804.0: 9, 1006.0: 9, 417.0:
9, 470.0: 9, 965.0: 9, 138.0: 9, 16.0: 9, 712.0: 9, 515.0: 9, 55.0: 9, 361.0: 9,
544.0: 9, 1040.0: 9, 1106.0: 9, 1204.0: 9, 414.0: 9, 929.0: 9, 430.0: 9, 370.0:
9, 1101.0: 9, 255.0: 9, 1161.0: 9, 556.0: 9, 900.0: 9, 631.0: 9, 885.0: 9,
985.0: 9, 542.0: 9, 502.0: 9, 568.0: 9, 377.0: 9, 32.0: 9, 1272.0: 9, 543.0: 9,
1136.0: 9, 634.0: 9, 950.0: 9, 742.0: 9, 432.0: 9, 912.0: 9, 135.0: 9, 590.0: 9,
116.0: 9, 247.0: 9, 878.0: 9, 78.0: 9, 961.0: 9, 549.0: 9, 518.0: 9, 213.0: 9,
672.0: 9, 570.0: 9, 1052.0: 9, 920.0: 9, 107.0: 9, 644.0: 9, 362.0: 9, 394.0: 9,
1081.0: 9, 1185.0: 9, 1169.0: 9, 1255.0: 9, 1097.0: 9, 875.0: 9, 1269.0: 9,
699.0: 9, 505.0: 9, 1134.0: 9, 163.0: 9, 1110.0: 9, 90.0: 9, 576.0: 9, 732.0: 9,
191.0: 9, 126.0: 9, 898.0: 9, 343.0: 9, 1091.0: 9, 1276.0: 9, 1200.0: 9, 142.0:
9, 434.0: 9, 418.0: 9, 495.0: 9, 1220.0: 9, 440.0: 9, 1083.0: 9, 144.0: 9,
404.0: 9, 14.0: 9, 1074.0: 9, 474.0: 9, 774.0: 9, 830.0: 9, 1037.0: 9, 984.0: 9,
690.0: 9, 679.0: 9, 561.0: 9, 780.0: 9, 155.0: 9, 262.0: 9, 70.0: 9, 857.0: 9,
1241.0: 9, 180.0: 9, 990.0: 9, 687.0: 9, 1034.0: 9, 317.0: 9, 1005.0: 9, 1235.0:
9, 335.0: 9, 783.0: 9, 1094.0: 9, 1010.0: 9, 1256.0: 9, 301.0: 9, 407.0: 9,
161.0: 9, 22.0: 9, 293.0: 9, 261.0: 9, 996.0: 9, 469.0: 9, 877.0: 9, 572.0: 9,
597.0: 9, 600.0: 9, 767.0: 9, 190.0: 9, 1226.0: 9, 437.0: 9, 778.0: 9, 243.0: 9,
494.0: 9, 782.0: 9, 123.0: 9, 1178.0: 9, 201.0: 9, 279.0: 9, 533.0: 9, 516.0: 9,
487.0: 9, 425.0: 9, 1223.0: 9, 511.0: 9, 1066.0: 9, 359.0: 9, 1116.0: 9, 695.0:
9, 97.0: 9, 755.0: 9, 496.0: 9, 891.0: 9, 194.0: 9, 409.0: 9, 785.0: 9, 466.0:
9, 1071.0: 9, 1059.0: 9, 928.0: 9, 471.0: 9, 773.0: 9, 887.0: 9, 647.0: 9,
731.0: 9, 13.0: 9, 1015.0: 9, 1151.0: 9, 970.0: 9, 357.0: 9, 140.0: 9, 422.0: 9,
557.0: 9, 1208.0: 9, 1146.0: 9, 490.0: 9, 79.0: 9, 1243.0: 9, 1253.0: 9, 1035.0:
9, 1096.0: 9, 727.0: 9, 364.0: 9, 765.0: 9, 248.0: 9, 1233.0: 9, 906.0: 9,
1107.0: 9, 1061.0: 9, 368.0: 9, 981.0: 9, 639.0: 9, 1259.0: 9, 46.0: 9, 583.0:
9, 759.0: 9, 1153.0: 9, 1212.0: 9, 1218.0: 9, 517.0: 9, 483.0: 9, 630.0: 9,
1158.0: 9, 945.0: 9, 271.0: 9, 373.0: 9, 758.0: 9, 1273.0: 9, 12.0: 9, 401.0: 9,
522.0: 9, 702.0: 9, 1137.0: 9, 577.0: 9, 1254.0: 9, 363.0: 9, 1252.0: 9, 1213.0:
9, 536.0: 9, 1156.0: 9, 792.0: 9, 128.0: 9, 10.0: 9, 735.0: 9, 973.0: 9, 498.0:
9, 753.0: 9, 1001.0: 9, 567.0: 9, 1180.0: 9, 256.0: 9, 278.0: 9, 249.0: 9,
372.0: 9, 447.0: 9, 598.0: 9, 701.0: 9, 1090.0: 9, 184.0: 9, 3.0: 9, 81.0: 9,
960.0: 9, 87.0: 9, 257.0: 9, 907.0: 9, 801.0: 9, 550.0: 9, 334.0: 9, 1117.0: 9,
812.0: 9, 1089.0: 9, 797.0: 9, 998.0: 9, 433.0: 9, 387.0: 9, 165.0: 9, 121.0: 9,
1170.0: 9, 501.0: 9, 853.0: 9, 843.0: 9, 847.0: 9, 689.0: 9, 212.0: 9, 913.0: 9,
294.0: 9, 1172.0: 9, 398.0: 9, 626.0: 9, 313.0: 9, 385.0: 9, 411.0: 9, 704.0: 9,
380.0: 9, 1182.0: 9, 509.0: 9, 848.0: 9, 750.0: 9, 277.0: 9, 209.0: 9, 821.0: 9,
746.0: 9, 150.0: 9, 336.0: 9, 814.0: 9, 530.0: 9, 745.0: 9, 1270.0: 9, 1011.0:
9, 436.0: 9, 29.0: 9, 220.0: 9, 680.0: 9, 669.0: 9, 899.0: 9, 214.0: 9, 1036.0:
9, 479.0: 9, 659.0: 9, 1148.0: 9, 796.0: 9, 889.0: 9, 383.0: 9, 1109.0: 9,
1008.0: 9, 1278.0: 9, 859.0: 9, 956.0: 9, 513.0: 9, 874.0: 9, 991.0: 9, 707.0:
9, 198.0: 9, 11.0: 9, 645.0: 9, 959.0: 9, 633.0: 9, 350.0: 9, 1181.0: 9, 673.0:
9, 816.0: 9, 714.0: 9, 1084.0: 9, 1230.0: 9, 1077.0: 9, 1064.0: 9, 450.0: 9,
610.0: 9, 298.0: 9, 1188.0: 9, 98.0: 9, 224.0: 9, 770.0: 9, 111.0: 9, 321.0: 9,
405.0: 9, 503.0: 9, 1131.0: 9, 1177.0: 9, 30.0: 9, 682.0: 9, 412.0: 9, 1167.0:
9, 461.0: 9, 519.0: 9, 291.0: 9, 413.0: 9, 1205.0: 9, 1047.0: 9, 166.0: 9,
1080.0: 9, 1147.0: 9, 20.0: 9, 747.0: 9, 940.0: 9, 1138.0: 9, 552.0: 9, 1168.0:
9, 1078.0: 9, 292.0: 9, 932.0: 9, 491.0: 9, 1087.0: 9, 451.0: 9, 1215.0: 9,
1070.0: 9, 327.0: 9, 823.0: 9, 159.0: 9, 231.0: 9, 977.0: 9, 1012.0: 9, 798.0:
9, 222.0: 9, 344.0: 9, 475.0: 9, 295.0: 9, 657.0: 9, 322.0: 9, 1152.0: 9, 17.0:
9, 1139.0: 9, 681.0: 9, 160.0: 9, 508.0: 9, 169.0: 9, 77.0: 9, 763.0: 9, 266.0:
9, 276.0: 9, 84.0: 9, 807.0: 9, 1048.0: 9, 986.0: 9, 91.0: 9, 1154.0: 9, 845.0:
9, 454.0: 9, 143.0: 9, 259.0: 9, 188.0: 9, 1119.0: 9, 764.0: 9, 1099.0: 9,
938.0: 9, 1261.0: 9, 713.0: 9, 846.0: 9, 40.0: 9, 652.0: 9, 1113.0: 9, 800.0: 9,
616.0: 9, 963.0: 9, 953.0: 9, 331.0: 9, 1165.0: 9, 615.0: 9, 141.0: 9, 120.0: 9,
922.0: 9, 455.0: 9, 915.0: 9, 332.0: 9, 72.0: 9, 514.0: 9, 56.0: 9, 805.0: 9,
910.0: 9, 958.0: 9, 272.0: 9, 158.0: 9, 1198.0: 9, 1045.0: 9, 178.0: 9, 151.0:
9, 296.0: 9, 1060.0: 9, 688.0: 9, 219.0: 9, 1190.0: 9, 106.0: 9, 204.0: 9,
1073.0: 9, 994.0: 9, 58.0: 9, 1281.0: 9, 429.0: 9, 347.0: 9, 886.0: 9, 96.0: 9,
85.0: 9, 374.0: 9, 649.0: 9, 325.0: 9, 1279.0: 9, 127.0: 9, 1214.0: 9, 1150.0:
9, 1054.0: 9, 1000.0: 9, 1277.0: 9, 779.0: 9, 738.0: 9, 1029.0: 9, 274.0: 9,
1125.0: 9, 757.0: 9, 441.0: 9, 698.0: 9, 896.0: 9, 775.0: 9, 1224.0: 9, 439.0:
9, 607.0: 9, 145.0: 9, 444.0: 9, 175.0: 9, 1038.0: 9, 971.0: 9, 867.0: 9,
1046.0: 9, 1175.0: 9, 132.0: 9, 456.0: 9, 834.0: 9, 467.0: 9, 627.0: 9, 1195.0:
9, 488.0: 9, 38.0: 9, 663.0: 9, 988.0: 9, 992.0: 9, 1247.0: 9, 547.0: 9, 302.0:
9, 1176.0: 9, 914.0: 9, 73.0: 9, 103.0: 9, 628.0: 9, 200.0: 9, 402.0: 9, 661.0:
9, 396.0: 9, 366.0: 9, 102.0: 9, 825.0: 9, 1088.0: 9, 1232.0: 9, 240.0: 9,
168.0: 9, 1.0: 9, 1067.0: 9, 176.0: 9, 869.0: 9, 818.0: 9, 251.0: 9, 890.0: 9,
939.0: 9, 861.0: 9, 716.0: 9, 942.0: 9, 1266.0: 9, 1039.0: 9, 676.0: 9, 481.0:
9, 389.0: 9, 9.0: 9, 558.0: 9, 174.0: 9, 1104.0: 9, 269.0: 9, 99.0: 9, 743.0: 9,
119.0: 9, 1262.0: 9, 290.0: 9, 136.0: 9, 1120.0: 9, 164.0: 9, 751.0: 9, 8.0: 9,
217.0: 9, 962.0: 9, 15.0: 9, 320.0: 9, 225.0: 9, 791.0: 9, 185.0: 9, 1009.0: 9,
925.0: 9, 7.0: 9, 1257.0: 9, 416.0: 9, 476.0: 9, 187.0: 9, 1082.0: 9, 587.0: 9,
999.0: 9, 152.0: 9, 305.0: 9, 92.0: 9, 173.0: 9, 228.0: 9, 286.0: 9, 1149.0: 9,
872.0: 9, 41.0: 9, 1108.0: 9, 53.0: 9, 177.0: 9, 4.0: 9, 1244.0: 9, 574.0: 9,
211.0: 9, 1098.0: 9, 395.0: 9, 167.0: 9, 95.0: 9, 453.0: 9, 1141.0: 9, 354.0: 9,
365.0: 9, 1102.0: 9, 310.0: 9, 356.0: 9, 795.0: 9, 668.0: 9, 399.0: 9, 966.0: 9,
223.0: 9, 50.0: 9, 667.0: 9, 864.0: 9, 691.0: 9, 888.0: 9, 893.0: 9, 849.0: 9,
1105.0: 9, 1017.0: 9, 54.0: 9, 1210.0: 9, 1201.0: 9, 876.0: 9, 1184.0: 9, 740.0:
9, 100.0: 9, 808.0: 9, 421.0: 9, 612.0: 9, 693.0: 9, 1207.0: 9, 239.0: 9,
1258.0: 9, 964.0: 9, 974.0: 9, 282.0: 9, 621.0: 9, 1197.0: 9, 1057.0: 9, 1020.0:
9, 288.0: 9, 153.0: 9, 118.0: 9, 553.0: 9, 186.0: 9, 504.0: 9, 1085.0: 9, 599.0:
9, 617.0: 9, 285.0: 9, 1155.0: 9, 108.0: 9, 1050.0: 9, 275.0: 9, 246.0: 9,
606.0: 9, 82.0: 9, 710.0: 9, 1023.0: 9, 304.0: 9, 263.0: 9, 826.0: 9, 419.0: 9,
975.0: 9, 33.0: 9, 793.0: 9, 1030.0: 9, 1143.0: 9, 711.0: 9, 392.0: 9, 895.0: 9,
1004.0: 9, 113.0: 9, 993.0: 9, 134.0: 9, 684.0: 9, 851.0: 9, 287.0: 9, 810.0: 9,
1014.0: 9, 579.0: 9, 1157.0: 9, 756.0: 9, 253.0: 9, 391.0: 9, 1142.0: 9, 130.0:
9, 806.0: 9, 1217.0: 9, 1196.0: 9, 39.0: 9, 312.0: 9, 1123.0: 9, 36.0: 9, 468.0:
9, 195.0: 9, 1028.0: 9, 1173.0: 9, 842.0: 9, 646.0: 9, 289.0: 9, 303.0: 9,
260.0: 9, 790.0: 9, 133.0: 9, 811.0: 9, 125.0: 9, 538.0: 9, 863.0: 9, 1043.0: 9,
35.0: 9, 486.0: 9})
```

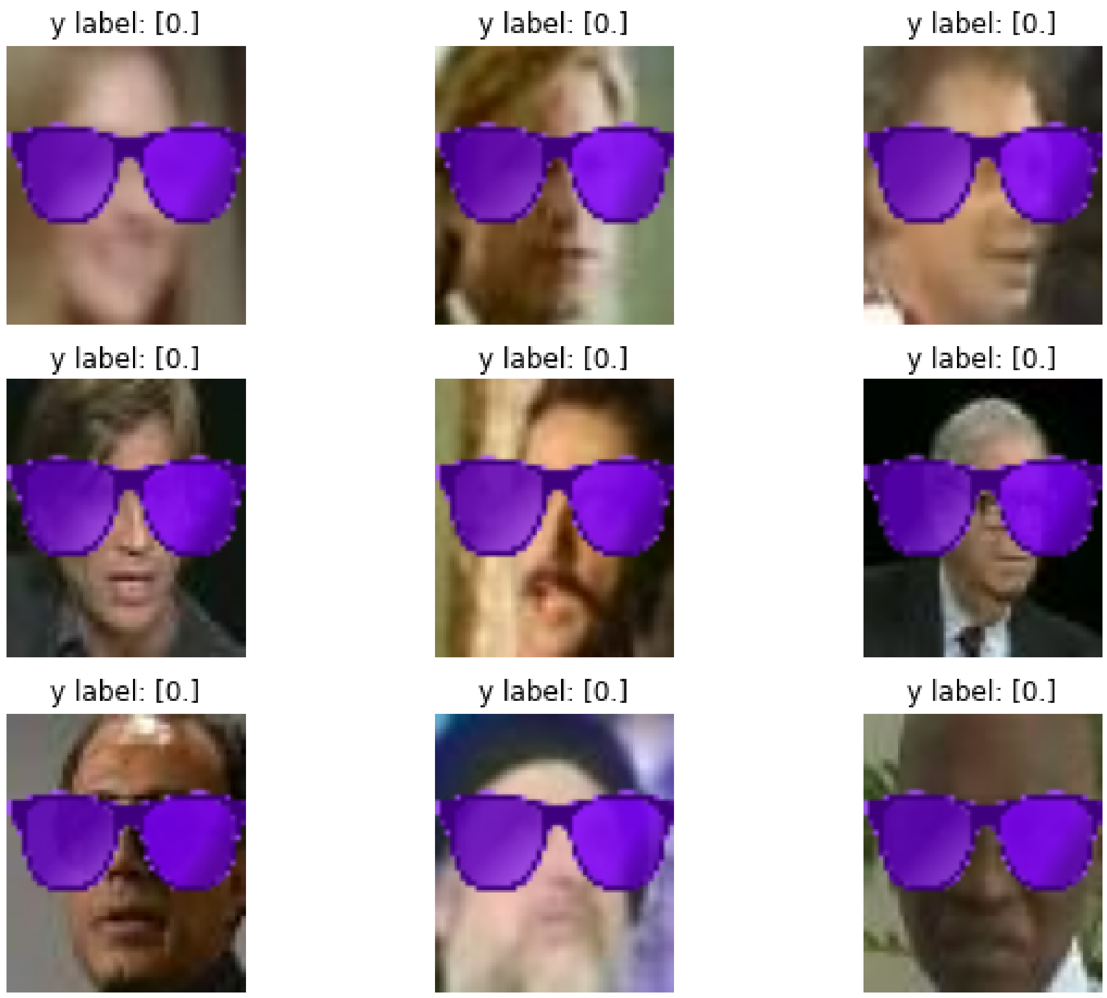
```
The length of the cl_valid dataset:  11547
The Number of classes in bd_valid dataset:  1
The Number of samples for each classes in bd_valid:  Counter({0.0: 11547})
The length of the bd_valid dataset:  11547

The Number of classes in cl_test dataset:  1283
The Number of samples for each classes in cl_test:  Counter({950: 10, 992: 10,
823: 10, 949: 10, 1274: 10, 788: 10, 287: 10, 241: 10, 200: 10, 860: 10, 108:
10, 1035: 10, 86: 10, 537: 10, 166: 10, 757: 10, 172: 10, 261: 10, 393: 10, 341:
10, 217: 10, 1174: 10, 255: 10, 903: 10, 75: 10, 297: 10, 1273: 10, 55: 10, 649:
10, 528: 10, 965: 10, 106: 10, 368: 10, 52: 10, 234: 10, 102: 10, 592: 10, 812:
10, 819: 10, 807: 10, 290: 10, 80: 10, 352: 10, 1098: 10, 754: 10, 832: 10, 46:
10, 1021: 10, 843: 10, 1278: 10, 1088: 10, 1081: 10, 398: 10, 1212: 10, 303: 10,
29: 10, 953: 10, 141: 10, 749: 10, 962: 10, 205: 10, 1186: 10, 857: 10, 33: 10,
610: 10, 885: 10, 1276: 10, 914: 10, 1077: 10, 1248: 10, 375: 10, 1180: 10, 95:
10, 354: 10, 1241: 10, 63: 10, 327: 10, 1039: 10, 1119: 10, 1028: 10, 793: 10,
529: 10, 770: 10, 1210: 10, 824: 10, 357: 10, 856: 10, 413: 10, 337: 10, 138:
10, 265: 10, 874: 10, 596: 10, 195: 10, 356: 10, 742: 10, 397: 10, 1279: 10,
710: 10, 745: 10, 1137: 10, 1122: 10, 704: 10, 1067: 10, 1076: 10, 1218: 10,
668: 10, 182: 10, 201: 10, 548: 10, 1166: 10, 573: 10, 277: 10, 1127: 10, 584:
10, 1105: 10, 815: 10, 838: 10, 1075: 10, 830: 10, 77: 10, 634: 10, 800: 10,
1281: 10, 1094: 10, 633: 10, 1106: 10, 889: 10, 460: 10, 329: 10, 736: 10, 655:
10, 145: 10, 239: 10, 308: 10, 491: 10, 642: 10, 386: 10, 1002: 10, 796: 10,
762: 10, 617: 10, 32: 10, 932: 10, 474: 10, 831: 10, 755: 10, 881: 10, 294: 10,
351: 10, 1258: 10, 266: 10, 192: 10, 228: 10, 602: 10, 214: 10, 68: 10, 873: 10,
154: 10, 1252: 10, 1136: 10, 333: 10, 307: 10, 562: 10, 448: 10, 618: 10, 929:
10, 1164: 10, 961: 10, 1201: 10, 1192: 10, 39: 10, 35: 10, 628: 10, 870: 10,
1107: 10, 88: 10, 215: 10, 498: 10, 175: 10, 414: 10, 1243: 10, 403: 10, 1069:
10, 387: 10, 1233: 10, 619: 10, 991: 10, 1202: 10, 959: 10, 740: 10, 1104: 10,
553: 10, 622: 10, 559: 10, 783: 10, 828: 10, 586: 10, 165: 10, 977: 10, 1259:
10, 257: 10, 1066: 10, 100: 10, 847: 10, 378: 10, 272: 10, 302: 10, 1030: 10,
486: 10, 64: 10, 872: 10, 1198: 10, 1061: 10, 229: 10, 973: 10, 531: 10, 591:
10, 661: 10, 556: 10, 765: 10, 1150: 10, 558: 10, 273: 10, 583: 10, 599: 10,
758: 10, 980: 10, 629: 10, 26: 10, 391: 10, 698: 10, 630: 10, 1217: 10, 1181:
10, 452: 10, 295: 10, 109: 10, 1247: 10, 135: 10, 1006: 10, 605: 10, 709: 10,
70: 10, 1008: 10, 269: 10, 1213: 10, 841: 10, 479: 10, 956: 10, 1057: 10, 527:
10, 844: 10, 806: 10, 723: 10, 1092: 10, 18: 10, 293: 10, 542: 10, 1142: 10,
1215: 10, 394: 10, 852: 10, 791: 10, 1250: 10, 1209: 10, 1124: 10, 868: 10, 415:
10, 853: 10, 464: 10, 779: 10, 1229: 10, 797: 10, 894: 10, 483: 10, 478: 10,
1249: 10, 637: 10, 377: 10, 543: 10, 263: 10, 130: 10, 51: 10, 10: 10, 705: 10,
854: 10, 19: 10, 1013: 10, 156: 10, 1172: 10, 405: 10, 85: 10, 951: 10, 1185:
10, 700: 10, 578: 10, 232: 10, 1157: 10, 939: 10, 680: 10, 1160: 10, 1123: 10,
66: 10, 703: 10, 199: 10, 1167: 10, 909: 10, 887: 10, 714: 10, 692: 10, 311: 10,
905: 10, 131: 10, 702: 10, 1029: 10, 927: 10, 1143: 10, 621: 10, 727: 10, 708:
10, 132: 10, 924: 10, 471: 10, 726: 10, 87: 10, 43: 10, 608: 10, 416: 10, 1031:
10, 246: 10, 224: 10, 772: 10, 990: 10, 871: 10, 418: 10, 372: 10, 675: 10, 399:
10, 1151: 10, 1044: 10, 571: 10, 59: 10, 207: 10, 884: 10, 971: 10, 908: 10,
672: 10, 669: 10, 1236: 10, 282: 10, 364: 10, 242: 10, 1163: 10, 69: 10, 766:
```

```
10, 615: 10, 248: 10, 817: 10, 406: 10, 667: 10, 739: 10, 968: 10, 1153: 10,
1282: 10, 612: 10, 202: 10, 233: 10, 671: 10, 1058: 10, 381: 10, 682: 10, 49:
10, 1226: 10, 313: 10, 113: 10, 1165: 10, 423: 10, 93: 10, 1018: 10, 981: 10,
283: 10, 1237: 10, 1203: 10, 279: 10, 942: 10, 314: 10, 223: 10, 829: 10, 442:
10, 839: 10, 227: 10, 557: 10, 181: 10, 893: 10, 1223: 10, 358: 10, 1265: 10,
688: 10, 430: 10, 99: 10, 437: 10, 1093: 10, 550: 10, 349: 10, 768: 10, 369: 10,
631: 10, 97: 10, 435: 10, 771: 10, 784: 10, 1085: 10, 1193: 10, 607: 10, 1220:
10, 886: 10, 331: 10, 769: 10, 6: 10, 994: 10, 988: 10, 506: 10, 252: 10, 922:
10, 1135: 10, 1171: 10, 408: 10, 183: 10, 1000: 10, 907: 10, 438: 10, 701: 10,
1197: 10, 879: 10, 808: 10, 421: 10, 445: 10, 171: 10, 967: 10, 250: 10, 850:
10, 760: 10, 335: 10, 748: 10, 970: 10, 555: 10, 57: 10, 810: 10, 513: 10, 105:
10, 1277: 10, 878: 10, 296: 10, 1016: 10, 1147: 10, 447: 10, 159: 10, 624: 10,
1225: 10, 1034: 10, 281: 10, 581: 10, 286: 10, 481: 10, 161: 10, 1221: 10, 896:
10, 208: 10, 136: 10, 580: 10, 1125: 10, 589: 10, 821: 10, 38: 10, 409: 10, 522:
10, 62: 10, 436: 10, 963: 10, 27: 10, 1084: 10, 858: 10, 834: 10, 775: 10, 155:
10, 521: 10, 936: 10, 876: 10, 1001: 10, 753: 10, 681: 10, 1149: 10, 1015: 10,
699: 10, 773: 10, 1275: 10, 767: 10, 875: 10, 179: 10, 928: 10, 1205: 10, 958:
10, 1062: 10, 1230: 10, 497: 10, 30: 10, 729: 10, 998: 10, 243: 10, 1120: 10,
433: 10, 367: 10, 107: 10, 730: 10, 846: 10, 1244: 10, 212: 10, 1065: 10, 8: 10,
835: 10, 1096: 10, 1256: 10, 1238: 10, 191: 10, 90: 10, 274: 10, 1033: 10, 1110:
10, 1004: 10, 72: 10, 422: 10, 449: 10, 501: 10, 176: 10, 158: 10, 1071: 10,
1131: 10, 309: 10, 1012: 10, 920: 10, 41: 10, 119: 10, 1040: 10, 129: 10, 645:
10, 361: 10, 560: 10, 92: 10, 547: 10, 1253: 10, 288: 10, 83: 10, 842: 10, 1263:
10, 715: 10, 695: 10, 995: 10, 1246: 10, 561: 10, 731: 10, 492: 10, 923: 10, 0:
10, 855: 10, 1109: 10, 979: 10, 913: 10, 590: 10, 380: 10, 1272: 10, 504: 10,
237: 10, 1130: 10, 676: 10, 22: 10, 382: 10, 1234: 10, 1176: 10, 840: 10, 746:
10, 417: 10, 477: 10, 814: 10, 1114: 10, 721: 10, 899: 10, 209: 10, 996: 10,
662: 10, 694: 10, 142: 10, 1043: 10, 1056: 10, 677: 10, 643: 10, 940: 10, 816:
10, 133: 10, 188: 10, 650: 10, 752: 10, 997: 10, 407: 10, 196: 10, 759: 10, 424:
10, 494: 10, 1048: 10, 226: 10, 551: 10, 1133: 10, 813: 10, 697: 10, 910: 10,
960: 10, 861: 10, 1024: 10, 56: 10, 1121: 10, 499: 10, 89: 10, 292: 10, 157: 10,
446: 10, 1268: 10, 125: 10, 122: 10, 451: 10, 472: 10, 300: 10, 890: 10, 606:
10, 818: 10, 601: 10, 1128: 10, 751: 10, 493: 10, 931: 10, 459: 10, 371: 10,
851: 10, 267: 10, 845: 10, 463: 10, 1214: 10, 383: 10, 925: 10, 146: 10, 326:
10, 663: 10, 776: 10, 712: 10, 1251: 10, 603: 10, 426: 10, 220: 10, 338: 10,
514: 10, 983: 10, 3: 10, 103: 10, 170: 10, 627: 10, 577: 10, 276: 10, 733: 10,
837: 10, 785: 10, 151: 10, 719: 10, 247: 10, 440: 10, 804: 10, 673: 10, 1188:
10, 999: 10, 473: 10, 582: 10, 706: 10, 111: 10, 532: 10, 641: 10, 379: 10, 518:
10, 1190: 10, 505: 10, 366: 10, 489: 10, 1139: 10, 1038: 10, 639: 10, 1026: 10,
539: 10, 1068: 10, 150: 10, 579: 10, 76: 10, 190: 10, 254: 10, 1051: 10, 187:
10, 891: 10, 1086: 10, 126: 10, 304: 10, 480: 10, 517: 10, 47: 10, 1116: 10,
1224: 10, 801: 10, 653: 10, 268: 10, 902: 10, 385: 10, 609: 10, 833: 10, 285:
10, 1020: 10, 238: 10, 469: 10, 541: 10, 21: 10, 1108: 10, 121: 10, 169: 10,
147: 10, 711: 10, 1115: 10, 895: 10, 1118: 10, 98: 10, 251: 10, 400: 10, 439:
10, 1155: 10, 646: 10, 376: 10, 597: 10, 1152: 10, 888: 10, 127: 10, 1242: 10,
546: 10, 1207: 10, 1079: 10, 774: 10, 1162: 10, 686: 10, 1199: 10, 1074: 10,
728: 10, 1037: 10, 485: 10, 725: 10, 919: 10, 683: 10, 346: 10, 1053: 10, 716:
10, 168: 10, 198: 10, 319: 10, 490: 10, 552: 10, 110: 10, 1047: 10, 623: 10,
428: 10, 825: 10, 1064: 10, 1112: 10, 519: 10, 820: 10, 544: 10, 53: 10, 363:
10, 756: 10, 1261: 10, 444: 10, 1170: 10, 836: 10, 993: 10, 864: 10, 419: 10,
1169: 10, 1046: 10, 240: 10, 348: 10, 218: 10, 432: 10, 1097: 10, 396: 10, 900:
10, 262: 10, 1159: 10, 937: 10, 1023: 10, 508: 10, 45: 10, 468: 10, 659: 10,
1189: 10, 948: 10, 193: 10, 219: 10, 1231: 10, 1187: 10, 74: 10, 461: 10, 897:
10, 1184: 10, 750: 10, 429: 10, 1132: 10, 652: 10, 457: 10, 978: 10, 781: 10,
360: 10, 456: 10, 321: 10, 65: 10, 453: 10, 144: 10, 1168: 10, 611: 10, 1087:
10, 465: 10, 1073: 10, 822: 10, 691: 10, 614: 10, 503: 10, 975: 10, 918: 10,
1219: 10, 569: 10, 777: 10, 941: 10, 153: 10, 178: 10, 343: 10, 390: 10, 1183:
10, 40: 10, 1179: 10, 177: 10, 148: 10, 284: 10, 644: 10, 270: 10, 917: 10, 502:
10, 625: 10, 42: 10, 1266: 10, 568: 10, 28: 10, 402: 10, 1052: 10, 585: 10, 500:
10, 1059: 10, 427: 10, 593: 10, 163: 10, 604: 10, 488: 10, 722: 10, 1101: 10,
713: 10, 189: 10, 316: 10, 880: 10, 877: 10, 322: 10, 1262: 10, 1140: 10, 167:
10, 1134: 10, 101: 10, 724: 10, 1082: 10, 594: 10, 11: 10, 365: 10, 16: 10,
1204: 10, 1007: 10, 14: 10, 91: 10, 260: 10, 566: 10, 355: 10, 1103: 10, 545:
10, 934: 10, 37: 10, 25: 10, 795: 10, 1032: 10, 225: 10, 507: 10, 538: 10, 15:
10, 911: 10, 347: 10, 280: 10, 938: 10, 1182: 10, 1227: 10, 1161: 10, 865: 10,
866: 10, 509: 10, 476: 10, 587: 10, 120: 10, 1196: 10, 78: 10, 588: 10, 567: 10,
206: 10, 869: 10, 116: 10, 1146: 10, 1049: 10, 572: 10, 1264: 10, 1080: 10,
1060: 10, 278: 10, 1014: 10, 137: 10, 1267: 10, 1271: 10, 933: 10, 1099: 10,
332: 10, 370: 10, 24: 10, 342: 10, 720: 10, 204: 10, 1141: 10, 1194: 10, 1200:
10, 626: 10, 197: 10, 244: 10, 787: 10, 330: 10, 104: 10, 299: 10, 162: 10, 222:
10, 1260: 10, 660: 10, 534: 10, 1239: 10, 325: 10, 985: 10, 916: 10, 969: 10,
1090: 10, 616: 10, 48: 10, 275: 10, 734: 10, 34: 10, 235: 10, 350: 10, 67: 10,
236: 10, 523: 10, 344: 10, 253: 10, 384: 10, 862: 10, 665: 10, 510: 10, 79: 10,
475: 10, 1145: 10, 441: 10, 1083: 10, 678: 10, 23: 10, 648: 10, 310: 10, 763:
10, 1019: 10, 345: 10, 362: 10, 690: 10, 613: 10, 13: 10, 827: 10, 798: 10, 685:
10, 230: 10, 467: 10, 635: 10, 575: 10, 1269: 10, 289: 10, 184: 10, 651: 10,
986: 10, 892: 10, 315: 10, 849: 10, 952: 10, 1138: 10, 5: 10, 328: 10, 82: 10,
982: 10, 462: 10, 747: 10, 1055: 10, 1254: 10, 1111: 10, 826: 10, 12: 10, 1072:
10, 525: 10, 1050: 10, 216: 10, 1091: 10, 1041: 10, 564: 10, 334: 10, 320: 10,
1245: 10, 117: 10, 859: 10, 802: 10, 638: 10, 657: 10, 470: 10, 258: 10, 245:
10, 139: 10, 353: 10, 412: 10, 600: 10, 1178: 10, 964: 10, 395: 10, 926: 10,
1158: 10, 173: 10, 883: 10, 882: 10, 520: 10, 1: 10, 454: 10, 484: 10, 744: 10,
495: 10, 654: 10, 1235: 10, 647: 10, 1017: 10, 487: 10, 81: 10, 50: 10, 540: 10,
516: 10, 684: 10, 160: 10, 1216: 10, 1228: 10, 466: 10, 1175: 10, 410: 10, 458:
10, 1129: 10, 738: 10, 530: 10, 140: 10, 943: 10, 323: 10, 957: 10, 152: 10,
336: 10, 511: 10, 972: 10, 231: 10, 512: 10, 7: 10, 264: 10, 1100: 10, 656: 10,
863: 10, 906: 10, 1025: 10, 1195: 10, 44: 10, 317: 10, 554: 10, 632: 10, 674:
10, 305: 10, 71: 10, 732: 10, 373: 10, 1257: 10, 4: 10, 563: 10, 735: 10, 249:
10, 1070: 10, 1003: 10, 221: 10, 389: 10, 210: 10, 803: 10, 1211: 10, 431: 10,
598: 10, 271: 10, 717: 10, 174: 10, 1011: 10, 324: 10, 901: 10, 987: 10, 805:
10, 425: 10, 1191: 10, 693: 10, 848: 10, 1078: 10, 1095: 10, 670: 10, 1173: 10,
1054: 10, 955: 10, 1036: 10, 411: 10, 301: 10, 921: 10, 764: 10, 9: 10, 213: 10,
312: 10, 54: 10, 620: 10, 904: 10, 2: 10, 741: 10, 94: 10, 689: 10, 388: 10,
954: 10, 1148: 10, 359: 10, 1222: 10, 524: 10, 194: 10, 974: 10, 1232: 10, 123:
10, 1027: 10, 794: 10, 707: 10, 185: 10, 1255: 10, 404: 10, 947: 10, 1208: 10,
118: 10, 60: 10, 790: 10, 180: 10, 533: 10, 664: 10, 1113: 10, 944: 10, 898: 10,
778: 10, 666: 10, 526: 10, 36: 10, 718: 10, 1045: 10, 149: 10, 549: 10, 392: 10,
1154: 10, 792: 10, 1280: 10, 211: 10, 915: 10, 782: 10, 128: 10, 96: 10, 1206:
10, 17: 10, 945: 10, 1089: 10, 61: 10, 318: 10, 1117: 10, 679: 10, 535: 10, 58:
10, 482: 10, 930: 10, 443: 10, 1240: 10, 143: 10, 640: 10, 576: 10, 434: 10,
450: 10, 401: 10, 574: 10, 789: 10, 84: 10, 1126: 10, 935: 10, 1144: 10, 114:
10, 115: 10, 20: 10, 658: 10, 565: 10, 1005: 10, 186: 10, 743: 10, 737: 10, 780:
10, 339: 10, 298: 10, 73: 10, 203: 10, 496: 10, 1042: 10, 811: 10, 687: 10, 946:
10, 966: 10, 164: 10, 112: 10, 989: 10, 984: 10, 515: 10, 1063: 10, 124: 10,
536: 10, 976: 10, 306: 10, 1009: 10, 1010: 10, 799: 10, 761: 10, 867: 10, 134:
10, 696: 10, 1270: 10, 570: 10, 31: 10, 786: 10, 1022: 10, 256: 10, 340: 10,
912: 10, 259: 10, 374: 10, 1156: 10, 1102: 10, 809: 10, 636: 10, 291: 10, 455:
10, 420: 10, 1177: 10, 595: 10})
The length of the cl_test dataset:  12830
The Number of classes in bd_test dataset:  1
The Number of samples for each classes in bd_test:  Counter({0: 12830})
The length of the bd_test dataset:  12830
```

Now, let us see some examples for each classes in each dataset

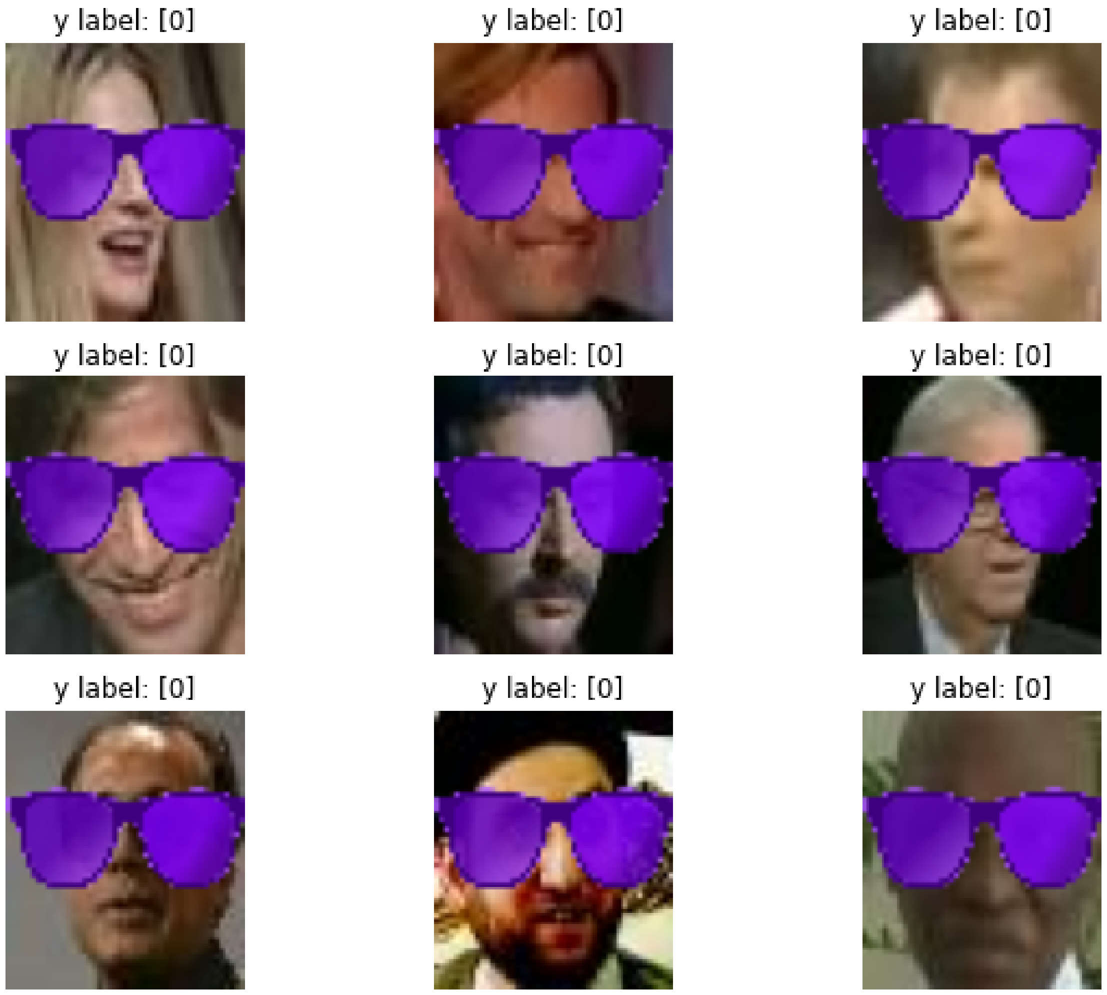
[36]:
```
# Examples from cl_valid dataset
figure = plt.figure(figsize=(10,8))
cols, rows = 3,3
for i in range(1, cols*rows+1):
  index = np.argwhere(y_cl_valid == i - 1)
  img, label = (x_cl_valid[index[0]], y_cl_valid[index[0]])
  figure.add_subplot(rows, cols, i)
  plt.title("y label: {}".format(label))
  plt.axis("off")
  plt.imshow(img[0]/255)
plt.show()
```

[37]:
```
# Examples from bd_valid dataset
figure = plt.figure(figsize=(10,8))
cols, rows = 3,3
for i in range(1, cols*rows+1):
  index = np.argwhere(y_cl_valid == i - 1)
  img, label = (x_bd_valid[index[0]], y_bd_valid[index[0]])
  figure.add_subplot(rows, cols, i)
  plt.title("y label: {}".format(label))
  plt.axis("off")
  plt.imshow(img[0]/255)
plt.show()
```



[38]:
```python
# Examples from cl_test dataset
figure = plt.figure(figsize=(10,8))
cols, rows = 3,3
for i in range(1, cols*rows+1):
  index = np.argwhere(y_cl_test == i - 1)
  img, label = (x_cl_test[index[0]], y_cl_test[index[0]])
  figure.add_subplot(rows, cols, i)
  plt.title("y label: {}".format(label))
  plt.axis("off")
  plt.imshow(img[0]/255)
plt.show()
```

[40]:
```python
# Examples from bd_test dataset
figure = plt.figure(figsize=(10,8))
cols, rows = 3,3
for i in range(1, cols*rows+1):
  index = np.argwhere(y_cl_test == i - 1)
  img, label = (x_bd_test[index[0]], y_bd_test[index[0]])
  figure.add_subplot(rows, cols, i)
  plt.title("y label: {}".format(label))
  plt.axis("off")
  plt.imshow(img[0]/255)
plt.show()
```



Conclusions from data and examples above:

- The clean and labelled dataset has 1283 classes, validation dataset has 9 samples for each class, and test dataset has 10 samples for each class.
- The each sample in bd dataset is transformed from cl dataset by adding sunglasses to each image, all of them will activates the backdoor for backdoored network. The backdoored network will classified all of them to class 0.
- The idea of lab is to use pruning defense to prune the channels in increasing order of their average activation values to create a new model B' which will have the ability to classify images with sunglasses to any other classes other than 0 and try our best to keep the clean classification accuraccy as large as possible.

1.5 Create new model B' and G using the pruning defense

[10]:
```
# Design the G model
class GoodNet_Model(keras.Model):
  def __init__(self, BadNet, BadNet_prime):
    super(GoodNet_Model, self).__init__()
    self.BadNet = BadNet
    self.BadNet_prime = BadNet_prime

  def predict(self, dataset):
    y = np.argmax(self.BadNet(dataset), axis=1)
    y_prime = np.argmax(self.BadNet_prime(dataset), axis=1)
    pred = []
    for i in range(dataset.shape[0]):
      if y[i]==y_prime[i]:
        pred.append(y[i])
      else:
        pred.append(1283)
    return pred
```

[11]:
```
# Clone the BadNet for pruning
BadNet_prime = keras.models.clone_model(BadNet)
BadNet_prime.set_weights(BadNet.get_weights())
```

[12]:
```
# Sort the average activation values in decreasing order and get the channel␣
↪indexes of them
pooling_layer_output = BadNet.get_layer('pool_3').output
# Cut off the model and make predictions for calculating average activation␣
↪values
cutoff_model=keras.models.Model(inputs=BadNet.
↪input,outputs=pooling_layer_output)
cutoff_predict = cutoff_model.predict(x_cl_valid)
print(cutoff_model.summary())
```

```
Model: "model"
_________________________________________________________________
 Layer (type)                Output Shape              Param #
=================================================================
 input (InputLayer)          [(None, 55, 47, 3)]       0

 conv_1 (Conv2D)             (None, 52, 44, 20)        980

 pool_1 (MaxPooling2D)       (None, 26, 22, 20)        0

 conv_2 (Conv2D)             (None, 24, 20, 40)        7240

 pool_2 (MaxPooling2D)       (None, 12, 10, 40)        0
 conv_3 (Conv2D)             (None, 10, 8, 60)         21660


 pool_3 (MaxPooling2D)       (None, 5, 4, 60)          0

=================================================================
Total params: 29,880
Trainable params: 29,880
Non-trainable params: 0
_________________________________________________________________
None
```

[13]:
```
# calculate the average activation values and sort their channel indexes from␣
 ↪small to large
average_activation_values = np.mean(cutoff_predict, axis=(0,1,2))
print("The average activation values before sorting:")
print(average_activation_values)
# Sort the average activation values in increasing order
sorted_index = np.argsort(average_activation_values)
print("The average activation values after sorting:")
print(average_activation_values[sorted_index])
print("The channel indexes sorting according to average activation values: ")
print(sorted_index)
```

```
The average activation values before sorting:
[0.0000000e+00 8.5787827e-01 0.0000000e+00 5.3079766e-01 5.1451387e+00
 2.0289593e+00 6.2408652e-03 5.3690352e+00 2.1106052e+00 0.0000000e+00
 4.1488919e+00 2.1980383e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 1.5654893e+00 0.0000000e+00 5.0868425e+00 2.4381560e-01
 1.8378238e-01 8.3539158e-02 4.3979712e-02 3.0290736e-03 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 4.8440866e+00 1.0589786e+00
 0.0000000e+00 0.0000000e+00 1.5006671e-02 0.0000000e+00 0.0000000e+00
 4.8648095e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 5.7658595e-01 4.2763174e-01 0.0000000e+00
 0.0000000e+00 1.8540380e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 1.3321568e-02 6.2038713e+00 0.0000000e+00 3.6191165e+00
 0.0000000e+00 1.6352931e+00 8.2229853e+00 5.0732869e-01 0.0000000e+00]
The average activation values after sorting:
[0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00 0.0000000e+00
 0.0000000e+00 3.0290736e-03 6.2408652e-03 1.3321568e-02 1.5006671e-02
 4.3979712e-02 8.3539158e-02 1.8378238e-01 2.4381560e-01 4.2763174e-01
 5.0732869e-01 5.3079766e-01 5.7658595e-01 8.5787827e-01 1.0589786e+00
 1.5654893e+00 1.6352931e+00 1.8540380e+00 2.0289593e+00 2.1106052e+00
 2.1980383e+00 3.6191165e+00 4.1488919e+00 4.8440866e+00 4.8648095e+00
 5.0868425e+00 5.1451387e+00 5.3690352e+00 6.2038713e+00 8.2229853e+00]
The channel indexes sorting according to average activation values:
[ 0 26 27 30 31 33 34 36 37 38 25 39 41 44 45 47 48 49 50 53 55 40 24 59
  9  2 12 13 17 14 15 23  6 51 32 22 21 20 19 43 58  3 42  1 29 16 56 46
  5  8 11 54 10 28 35 18  4  7 52 57]
```

[44]:
```
# Origin clean_classification_accuracy
clean_classification_accuracy = 98.64899974019225
# Prepare lists for data saving
clean_classification_acc = []
attack_success_rate = []
# Create flags for saving the models only once
save_2_model_flag = False
save_4_model_flag = False
save_10_model_flag = False
save_30_model_flag = False
# As "pool_3" layer do not have parameters for us to prune, we will get␣
↪"conv_3" layer's weight for pruning
weights_conv3 = BadNet.get_layer('conv_3').get_weights()[0]
bias_conv3 = BadNet.get_layer('conv_3').get_weights()[1]

# Pruning defense according to valid dataset, evaluate model using test␣
↪dataset, and save data for ploting
for index, channel_index in enumerate(sorted_index):
  start_time = time.perf_counter()
  # Pruning the channel by setting its weights to 0
  weights_conv3[:,:,:,channel_index] = 0
  bias_conv3[channel_index] = 0
  BadNet_prime.get_layer('conv_3').set_weights([weights_conv3, bias_conv3])
  # Combine B and B' to G model
  GoodNet = GoodNet_Model(BadNet, BadNet_prime)
  # Copy the code in eval.py for calculation and evaluation
  # Calculate clear acc using valid dataset for saving model
  cl_valid_pred = np.argmax(BadNet_prime.predict(x_cl_valid), axis = 1)
  cl_valid_acc = np.mean(np.equal(cl_valid_pred, y_cl_valid))*100
  # Calculate the clear acc using test dataset for ploting
  cl_test_pred = GoodNet.predict(x_cl_test)
  cl_test_acc = np.mean(np.equal(cl_test_pred, y_cl_test))*100
  clean_classification_acc.append(cl_test_acc)
  # Calculate the attack success rate using test dataset for ploting
  bd_test_pred = GoodNet.predict(x_bd_test)
  asr = np.mean(np.equal(bd_test_pred, y_bd_test))*100
  attack_success_rate.append(asr)
  stop_time = time.perf_counter()
  # Paint data
  print("The %d/%d time of pruning: "%((index+1), len(sorted_index)))
  print("The pruning channel index is: %d"%channel_index)
  print("The clean classification accuracy is: %f"%cl_test_acc)
  print("The attack success rate is: %f"%asr)
  print("The costing time: %f"%(stop_time - start_time))
  # Saving B' model for evaluation
  if clean_classification_accuracy - cl_valid_acc >= 2 and save_2_model_flag ==␣
↪False:
    print("The accuracy drops at least 2%, saving B' model")
    BadNet_prime.save('MLSecurity-Lab3/lab3/models/bd_net_prime_2.h5')
    save_2_model_flag = True
  if clean_classification_accuracy - cl_valid_acc >= 4 and save_4_model_flag ==␣
↪False:
    print("The accuracy drops at least 4%, saving B' model")
    BadNet_prime.save('MLSecurity-Lab3/lab3/models/bd_net_prime_4.h5')
    save_4_model_flag = True
  if clean_classification_accuracy - cl_valid_acc >= 10 and save_10_model_flag␣
↪== False:
    print("The accuracy drops at least 10%, saving B' model")
    BadNet_prime.save('MLSecurity-Lab3/lab3/models/bd_net_prime_10.h5')
    save_10_model_flag = True
  if clean_classification_accuracy - cl_valid_acc >= 30 and save_30_model_flag␣
↪== False:
    print("The accuracy drops at least 30%, saving B' model")
    BadNet_prime.save('MLSecurity-Lab3/lab3/models/bd_net_prime_30.h5')
    save_30_model_flag = True
  keras.backend.clear_session()
```

```
The 1/60 time of pruning:
The pruning channel index is: 0
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 3.648601
The 2/60 time of pruning:
The pruning channel index is: 26
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.945307
The 3/60 time of pruning:
The pruning channel index is: 27
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.995059
The 4/60 time of pruning:
The pruning channel index is: 30
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 3.081186
The 5/60 time of pruning:
```

```
The pruning channel index is: 31
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.905513
The 6/60 time of pruning:
The pruning channel index is: 33
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.927661
The 7/60 time of pruning:
The pruning channel index is: 34
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.880793
The 8/60 time of pruning:
The pruning channel index is: 36
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.900818
The 9/60 time of pruning:
The pruning channel index is: 37
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 3.015275
The 10/60 time of pruning:
The pruning channel index is: 38
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.968202
The 11/60 time of pruning:
The pruning channel index is: 25
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.983352
The 12/60 time of pruning:
The pruning channel index is: 39
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 3.000369
The 13/60 time of pruning:
The pruning channel index is: 41
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 3.012509
The 14/60 time of pruning:
The pruning channel index is: 44
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 3.014323
The 15/60 time of pruning:
The pruning channel index is: 45
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.954155
The 16/60 time of pruning:
The pruning channel index is: 47
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.849466
The 17/60 time of pruning:
The pruning channel index is: 48
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.894767
The 18/60 time of pruning:
The pruning channel index is: 49
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.892713
The 19/60 time of pruning:
The pruning channel index is: 50
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.845065
The 20/60 time of pruning:
The pruning channel index is: 53
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.892513
The 21/60 time of pruning:
The pruning channel index is: 55
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.892494
The 22/60 time of pruning:
The pruning channel index is: 40
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.889841
The 23/60 time of pruning:
The pruning channel index is: 24
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.856287
The 24/60 time of pruning:
The pruning channel index is: 59
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.878734
The 25/60 time of pruning:
The pruning channel index is: 9
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.920557
The 26/60 time of pruning:
The pruning channel index is: 2
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.864488
The 27/60 time of pruning:
The pruning channel index is: 12
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.891698
The 28/60 time of pruning:
The pruning channel index is: 13
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.881959
The 29/60 time of pruning:
The pruning channel index is: 17
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.889800
The 30/60 time of pruning:
The pruning channel index is: 14
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.892541
The 31/60 time of pruning:
The pruning channel index is: 15
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.875928
The 32/60 time of pruning:
The pruning channel index is: 23
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.889671
The 33/60 time of pruning:
The pruning channel index is: 6
The clean classification accuracy is: 98.620421
The attack success rate is: 100.000000
The costing time: 2.884284
The 34/60 time of pruning:
The pruning channel index is: 51
The clean classification accuracy is: 98.612627
The attack success rate is: 100.000000
The costing time: 2.854932
The 35/60 time of pruning:
The pruning channel index is: 32
The clean classification accuracy is: 98.604832
The attack success rate is: 100.000000
The costing time: 2.860164
The 36/60 time of pruning:
The pruning channel index is: 22
The clean classification accuracy is: 98.597038
The attack success rate is: 100.000000
The costing time: 2.897107
The 37/60 time of pruning:
The pruning channel index is: 21
The clean classification accuracy is: 98.597038
The attack success rate is: 100.000000
The costing time: 2.875752
The 38/60 time of pruning:
The pruning channel index is: 20
The clean classification accuracy is: 98.573655
The attack success rate is: 100.000000
The costing time: 2.845643
The 39/60 time of pruning:
The pruning channel index is: 19
The clean classification accuracy is: 98.526890
The attack success rate is: 100.000000
The costing time: 2.882415
The 40/60 time of pruning:
The pruning channel index is: 43
The clean classification accuracy is: 98.441154
The attack success rate is: 100.000000
The costing time: 2.906813
The 41/60 time of pruning:
The pruning channel index is: 58
The clean classification accuracy is: 98.409977
The attack success rate is: 100.000000
The costing time: 2.861933
The 42/60 time of pruning:
The pruning channel index is: 3
The clean classification accuracy is: 98.113796
The attack success rate is: 100.000000
The costing time: 2.874273
The 43/60 time of pruning:
The pruning channel index is: 42
The clean classification accuracy is: 97.747467
The attack success rate is: 100.000000
The costing time: 2.876975
The 44/60 time of pruning:
The pruning channel index is: 1
The clean classification accuracy is: 97.505846
The attack success rate is: 100.000000
The costing time: 2.854464
The 45/60 time of pruning:
The pruning channel index is: 29
The clean classification accuracy is: 95.744349
The attack success rate is: 100.000000
The costing time: 2.869190
The accuracy drops at least 2%, saving B' model
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.

/usr/local/lib/python3.7/dist-packages/keras/engine/functional.py:1410:
CustomMaskWarning: Custom mask layers require a config and must override
get_config. When loading, the custom mask layer must be passed to the
custom_objects argument.
  layer_config = serialize_layer_fn(layer)

The 46/60 time of pruning:
The pruning channel index is: 16
The clean classification accuracy is: 95.346843
The attack success rate is: 99.976617
The costing time: 2.873150
The 47/60 time of pruning:
The pruning channel index is: 56
The clean classification accuracy is: 94.902572
The attack success rate is: 99.984412
The costing time: 2.922548
The 48/60 time of pruning:
The pruning channel index is: 46
The clean classification accuracy is: 92.127825
The attack success rate is: 99.984412
The costing time: 2.865678
The accuracy drops at least 4%, saving B' model
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
The 49/60 time of pruning:
The pruning channel index is: 5
The clean classification accuracy is: 91.582229
The attack success rate is: 99.984412
The costing time: 2.857418
The 50/60 time of pruning:
The pruning channel index is: 8
The clean classification accuracy is: 91.130164
The attack success rate is: 99.976617
The costing time: 2.834128
The 51/60 time of pruning:
The pruning channel index is: 11
The clean classification accuracy is: 89.680436
The attack success rate is: 80.646921
The costing time: 2.885448
The 52/60 time of pruning:
The pruning channel index is: 54
The clean classification accuracy is: 84.333593
The attack success rate is: 77.209665
The costing time: 2.867509
The accuracy drops at least 10%, saving B' model
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
The 53/60 time of pruning:
The pruning channel index is: 10
The clean classification accuracy is: 76.165238
The attack success rate is: 36.266563
The costing time: 2.862291
The 54/60 time of pruning:
The pruning channel index is: 28
The clean classification accuracy is: 54.676539
The attack success rate is: 6.960249
The costing time: 2.864849
The accuracy drops at least 30%, saving B' model
WARNING:tensorflow:Compiled the loaded model, but the compiled metrics have yet
to be built. `model.compile_metrics` will be empty until you train or evaluate
the model.
The 55/60 time of pruning:
The pruning channel index is: 35
The clean classification accuracy is: 27.069369
The attack success rate is: 0.420889
The costing time: 2.833584
The 56/60 time of pruning:
The pruning channel index is: 18
The clean classification accuracy is: 13.702260
The attack success rate is: 0.000000
The costing time: 2.851175
The 57/60 time of pruning:
The pruning channel index is: 4
The clean classification accuracy is: 6.562744
The attack success rate is: 0.000000
The costing time: 2.860540
The 58/60 time of pruning:
The pruning channel index is: 7
The clean classification accuracy is: 1.519875
The attack success rate is: 0.000000
The costing time: 2.865595
The 59/60 time of pruning:
The pruning channel index is: 52
The clean classification accuracy is: 0.646921
The attack success rate is: 0.000000
The costing time: 2.898690
The 60/60 time of pruning:
The pruning channel index is: 57
The clean classification accuracy is: 0.070148
The attack success rate is: 0.000000
The costing time: 2.853630
```

1.6 Evaluate Repaired networks for X={2%,4%,10%}

Using eval.py to evaluate repaired networks for X=2%

[4]:
```
!python3 MLSecurity-Lab3/lab3/eval.py MLSecurity-Lab3/lab3/data/cl/test.h5␣
↪MLSecurity-Lab3/lab3/data/bd/bd_test.h5 MLSecurity-Lab3/lab3/models/
↪bd_net_prime_2.h5
```

```
2021-12-16 01:00:06.377722: W
tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding
allow_growth setting because the TF_FORCE_GPU_ALLOW_GROWTH environment variable
is set. Original config value was 0.
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Clean Classification accuracy: 95.90023382696803
Attack Success Rate: 100.0
```

Using eval.py to evaluate repaired networks for X=4%

[5]:
```
!python3 MLSecurity-Lab3/lab3/eval.py MLSecurity-Lab3/lab3/data/cl/test.h5␣
↪MLSecurity-Lab3/lab3/data/bd/bd_test.h5 MLSecurity-Lab3/lab3/models/
↪bd_net_prime_4.h5
```

```
2021-12-16 01:00:28.059246: W
tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding
allow_growth setting because the TF_FORCE_GPU_ALLOW_GROWTH environment variable
is set. Original config value was 0.
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Clean Classification accuracy: 92.29150428682775
Attack Success Rate: 99.98441153546376
```

Using eval.py to evaluate repaired networks for X=10%

[6]:

```
!python3 MLSecurity-Lab3/lab3/eval.py MLSecurity-Lab3/lab3/data/cl/test.h5␣
↪MLSecurity-Lab3/lab3/data/bd/bd_test.h5 MLSecurity-Lab3/lab3/models/
↪bd_net_prime_10.h5
```

```
2021-12-16 01:00:41.339635: W
tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding
allow_growth setting because the TF_FORCE_GPU_ALLOW_GROWTH environment variable
is set. Original config value was 0.
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Clean Classification accuracy: 84.54403741231489
Attack Success Rate: 77.20966484801247
```

Using eval.py to evaluate repaired networks for X=30%

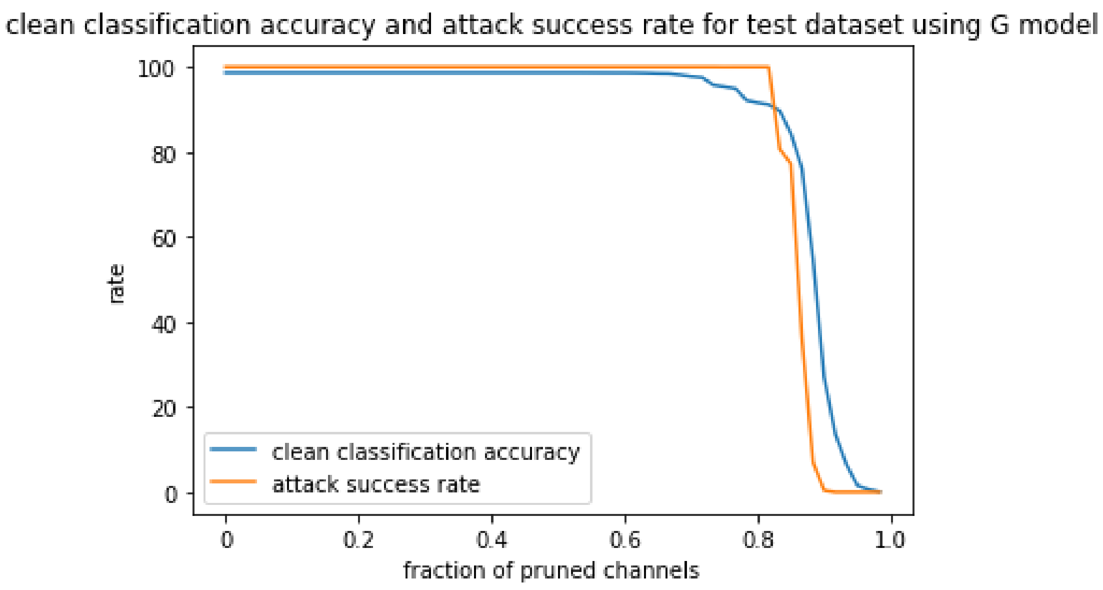
[4]:
```
!python3 MLSecurity-Lab3/lab3/eval.py MLSecurity-Lab3/lab3/data/cl/test.h5␣
↪MLSecurity-Lab3/lab3/data/bd/bd_test.h5 MLSecurity-Lab3/lab3/models/
↪bd_net_prime_30.h5
```

```
2021-12-16 03:58:12.710547: W
tensorflow/core/common_runtime/gpu/gpu_bfc_allocator.cc:39] Overriding
allow_growth setting because the TF_FORCE_GPU_ALLOW_GROWTH environment variable
is set. Original config value was 0.
WARNING:tensorflow:No training configuration found in the save file, so the
model was *not* compiled. Compile it manually.
Clean Classification accuracy: 54.762275915822286
Attack Success Rate: 6.96024941543258
```

Plot the accuracy on clean test data and the attack success rate (on backdoored test data) as a function of the fraction of channels pruned.

[43]:
```
plt.plot(clean_classification_acc)
plt.plot(attack_success_rate)
plt.xticks([0,12,24,36,48,60],[0,0.2,0.4,0.6,0.8,1.0])
plt.legend(['clean classification accuracy','attack success rate'])
plt.xlabel("fraction of pruned channels")
plt.ylabel("rate")
plt.title("clean classification accuracy and attack success rate for test␣
↪dataset using G model")
```

[43]:
```
Text(0.5, 1.0, 'clean classification accuracy and attack success rate for test
dataset using G model')
```



We can see in the plot above that though the G model can get the result which has zero attack success rate and non-zero clean classification accuracy, the result is not really successful. When the attack success rate decrease to zero, the clean classification accuracy is dropping over 50%. We can see from above that the clean classification accuracy begins to drop before the the attack success rate begins to drop which means that the reason of getting this result may be the bd_net we get is attacked using pruning aware attack.