Lab 6: Comparing White-Box Evasion Attacks

CSC 592: Machine Learning Security and Privacy

**Background**

There are a number of different white-box evasion attacks that have been developed in the adversarial machine learning literature. In today’s assignment we will experiment with the Fast Gradient Sign Method (FGSM), Projected Gradient Descent (PGD) attack and the Carlini and Wagner (C&W) attack. If you are interested the papers for each attack are given below (reading the papers is not needed for the lab):

FGSM: <https://arxiv.org/abs/1412.6572>

PGD: <https://arxiv.org/pdf/1706.06083>

C&W: <https://arxiv.org/abs/1608.04644>

**Step by Step Guide**

Follow the instructions carefully step by step:

**Step 1:** Create a new Python project called “LabComparingAttacks” using the Integrated Development Environment (IDE) of your choice (e.g., Visual Studio, Visual Studio Code, PyCharm).

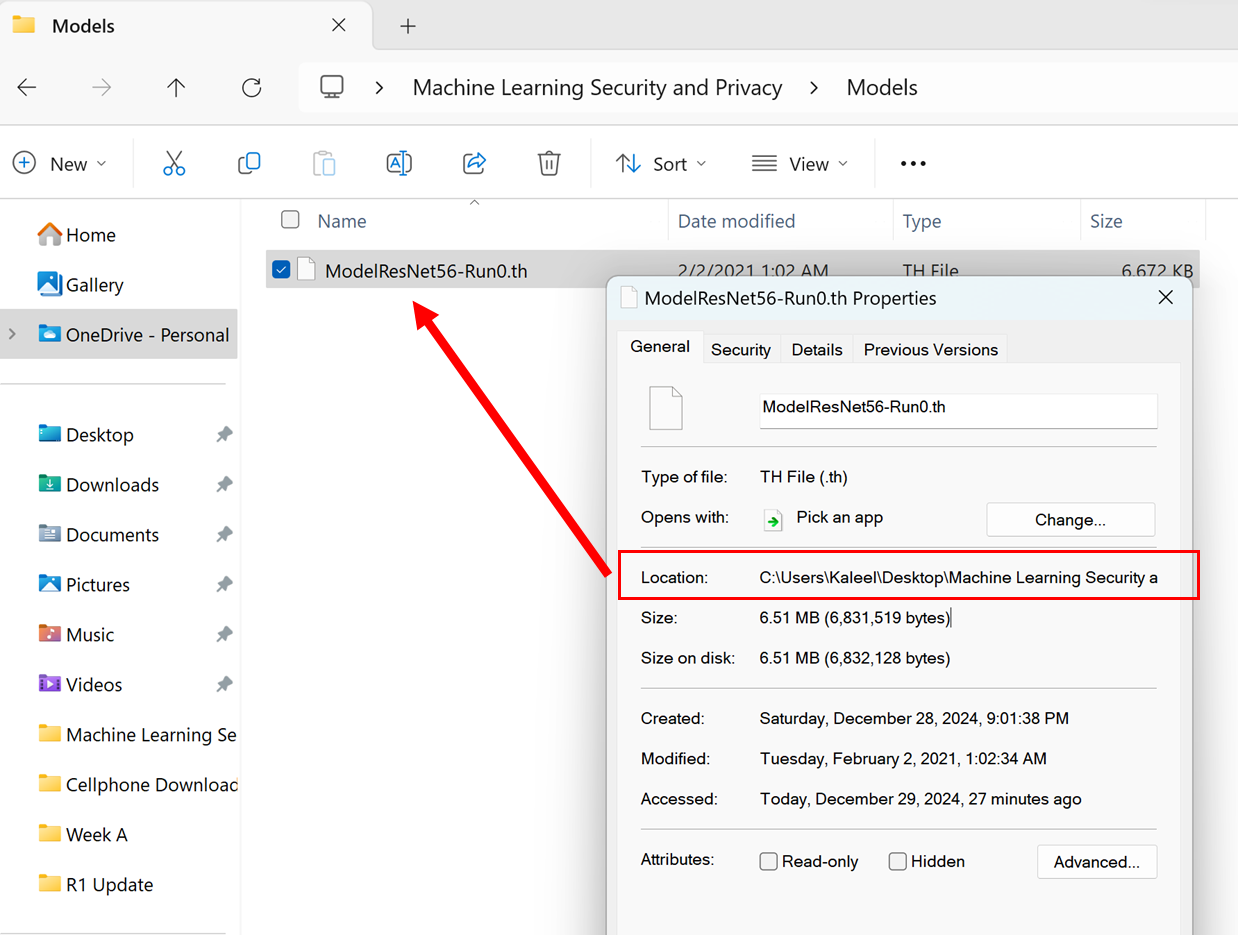
**Step 2:** Download and add three .py source files from Brightspace to your “LabComparingAttacks” Python project. The three source files are, “AttackWrappersWhiteBox.py”, “ResNet.py” and “DataManagerPytorchL.py”. Your project should contain the following files as shown in the screenshot below:

A screenshot of a computer

AI-generated content may be incorrect.

Note the “AttackWrappersWhiteBox.py” has been updated since the previous lab assignment. Make sure you download the latest version which contains the PGD and C&W attack methods. Likewise “DataManagerPytorchL.py” has also been updated so please make sure you download the latest version.

**Step 3:** Download the machine learning model we will be attacking from Brightspace. The model file is called “ModelResNet56-Run0.th”. Copy the file path of where the model is saved, as we will need it later. In Windows you can do this by right clicking on the model file, selecting properties, and then copying the value of the Location property. This is shown in the screenshot below:



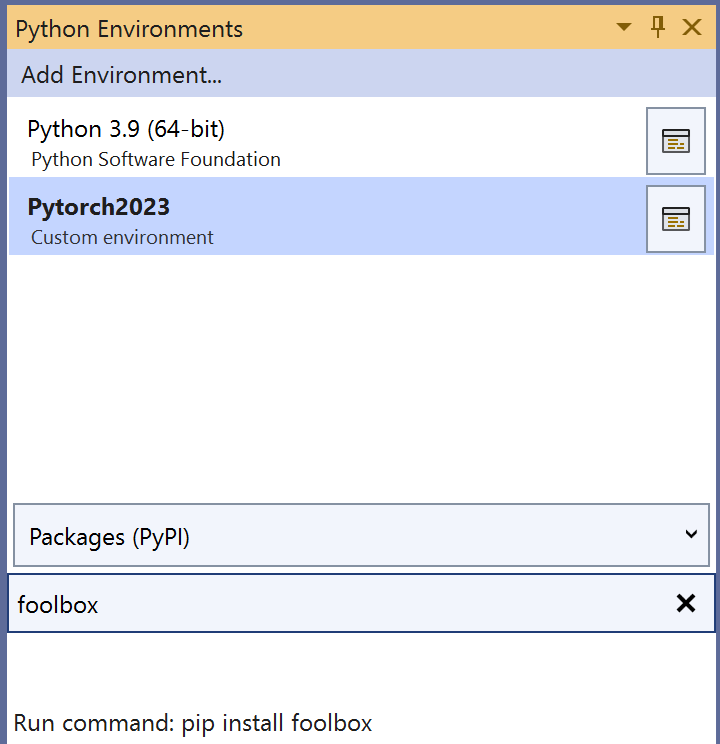
The ResNet is the same model from the lab 5, it has not changed.

**Step 4:** The Carlini and Wagner attack requires one additional library to run called Foolbox. In Visual Studio go to pip and run the command, “pip install foolbox” as shown below:

A screenshot of a computer

AI-generated content may be incorrect.

Click on “Manage Python Packages...” for the Python environment that you are currently working in.



Type foolbox in the search window and then click on “Run command: pip install foolbox”.

**Step 5:** Go to the “LabComparingAttacks.py” file and copy the following code:

import torch

import DataManagerPytorchL as DMP

import AttackWrappersWhiteBox

import ResNet

def main():

#Replace the next line with the file path of where you saved the ResNet model

modelDir = "C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th"

#Define the GPU device we are using

device = torch.device('cuda')

#Parameters for the dataset

batchSize = 64

numClasses = 10

inputImageSize = [1, 3, 32, 32] #Batch size, color channel, height, width

#Create the ResNet model (note this does not include pre-trained weights)

model = ResNet.resnet56(inputImageSize, numClasses).to(device)

#Next load in the trained weights of the model

checkpoint = torch.load(modelDir)

model.load\_state\_dict(checkpoint['state\_dict'])

#Switch the model into eval model for testing

model = model.eval()

#Load in the dataset

valLoader = DMP.GetCIFAR10Validation(inputImageSize[2], batchSize)

#Check the clean accuracy of the model

cleanAcc = DMP.validateD(valLoader, model, device)

print("CIFAR-10 Clean Val Loader Acc:", cleanAcc)

#Get correctly classified, classwise balanced samples to do the attack

totalSamplesRequired = 10

correctLoader = DMP.GetCorrectlyIdentifiedSamplesBalanced(model, totalSamplesRequired, valLoader, numClasses)

#Check to make sure the accuracy is 100% on the correct loader

correctAcc = DMP.validateD(correctLoader, model, device)

print("CIFAR-10 Clean Correct Loader Acc:", correctAcc)

#Do the attacks

epsilonMax = 0.031 #Maximum perturbation

clipMin = 0.0 #Minimum value a pixel can take

clipMax = 1.0 #Maximum value a pixel can take

numSteps = 10

epsilonStep = epsilonMax/numSteps

#Run the attacks

advLoader = AttackWrappersWhiteBox.FGSMNativePytorch(device, correctLoader, model, epsilonMax, clipMin, clipMax)

#advLoader = AttackWrappersWhiteBox.PGDNativePytorch(device, correctLoader, model, epsilonMax, epsilonStep, numSteps, clipMin, clipMax)

#advLoader = AttackWrappersWhiteBox.CWAttackUntargetedFoolBox(device, correctLoader, model, clipMin, clipMax)

advAcc = DMP.validateD(advLoader, model, device)

print("CIFAR-10 Adv Acc:", advAcc)

xCleanTensor, yCleaTensor = DMP.DataLoaderToTensor(correctLoader)

xAdvTensor, \_ = DMP.DataLoaderToTensor(advLoader)

DMP.ShowImages(xCleanTensor, xAdvTensor)

if \_\_name\_\_ == "\_\_main\_\_":

main()

**Step 6:** Recall that in step 3 you were asked to note down the file path of where you saved “ModelResNet56-Run0.th”. In the main function, replace the modelDir value with the file path where you saved “ModelResNet56-Run0.th”. For Professor Kaleel the file path looks like:

C://Users//Kaleel//Desktop//Machine Learning Security and Privacy//Models//ModelResNet56-Run0.th

**Step 7:** Read the code carefully to understand what is being run. The current code is set up to run FGSM on 10 clean, class-wise balanced, correctly identified samples from the CIFAR-10 validation dataset. The code then checks the class label that the ResNet-56 outputs for each adversarial example and reports the robustness. Lastly, the clean examples are displayed in the first row and the adversarial examples are displayed in the second row. Run the code. Your output should match what is displayed below:

A collage of animals

AI-generated content may be incorrect.

Note you may need to maximize the window to see larger versions of the images.

Files already downloaded and verified

CIFAR-10 Clean Val Loader Acc: 0.9277

CIFAR-10 Clean Correct Loader Acc: 1.0

Processing up to sample= 10

CIFAR-10 Adv Acc: 0.3

**Lab Assignment**

**Exercise 1:** Run the code with the FGSM attack and epsilonMax = 0.031. Record the output of the code (both the images and the output from the terminal).

**Exercise 2:** Run the PGD attack by commenting out the following line of code in “LabComparingAttacks”:

advLoader = AttackWrappersWhiteBox.FGSMNativePytorch(device, correctLoader, model, epsilonMax, clipMin, clipMax)

And uncommenting the following line:

#advLoader = AttackWrappersWhiteBox.PGDNativePytorch(device, correctLoader, model, epsilonMax, epsilonStep, numSteps, clipMin, clipMax)

Record the output of the code (both the images and the output from the terminal).

**Exercise 3:** Run the C&W attack by commenting out the following line of code in “LabComparingAttacks”:

advLoader = AttackWrappersWhiteBox.PGDNativePytorch(device, correctLoader, model, epsilonMax, epsilonStep, numSteps, clipMin, clipMax)

And uncommenting the following line:

#advLoader = AttackWrappersWhiteBox.CWAttackUntargetedFoolBox(device, correctLoader, model, clipMin, clipMax)

Note this attack may take some time to run. Record the output of the code (both the images and the output from the terminal).

**Exercise 4:** Visually inspect the images from the three attacks (FGSM, PGD and C&W). Which attack had the most visible perturbation? Which attack(s) were most effective?

**Exercise 5:** Repeat exercises 1-3 with epsilonMax = 0.062. What differences do you notice in terms of perturbation on the images and attack performance? Did the C&W results change between exercise 4 and exercise 5?

**Deliverables**

Submit the following two items on Brightspace:

Deliverable #1: A document containing the answers to exercises 1-5 and the corresponding code outputs.

Deliverable #2: A copy of your code (the .py files).