Amirkabir University of Technology
(Tehran Polytechnic)

Machine Learning

Final Project - Vision Group
VGG16

Supervisor
Dr. Sanaz Seyedin

By
Alireza Ansari

July 2023

Table of Contents

Amirkabir University of Technology
Faculty of Electrical Engineering

ML – Final Project

Chapter1: INTRODUCTION

A brief introduction of reviewed items in this phase.

Introduction

- VGG Net

  ✓ VGGNet was developed in 2014 by the **Visual Geometry Group** at Oxford University(hence the name VGG).

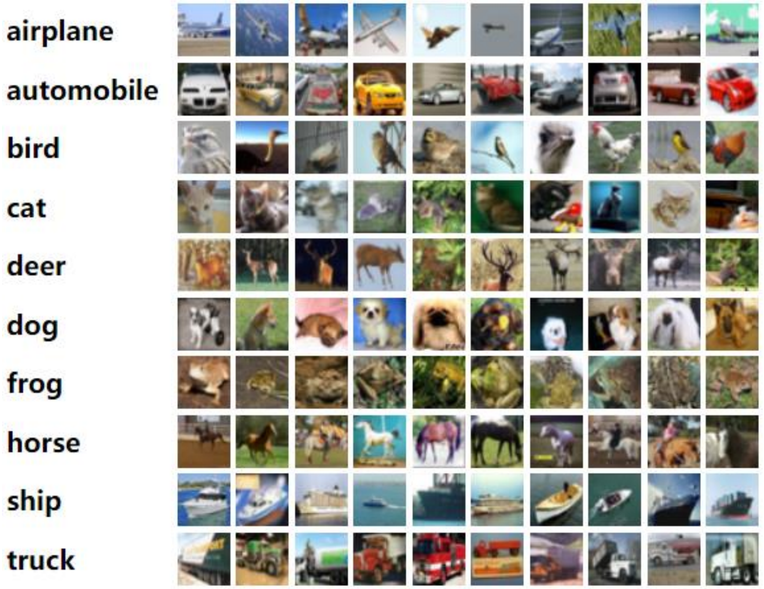
  ✓ The building components are exactly the same as those in **LeNet** and **AlexNet**, except that VGGNet is an even deeper network with more **convolutional**, **pooling**, and **dense layers**.

  ✓ VGGNet, also known as VGG16, consists of **16 weight layers**: **13 convolutional** layers and **3 fully connected** layers.

Introduction

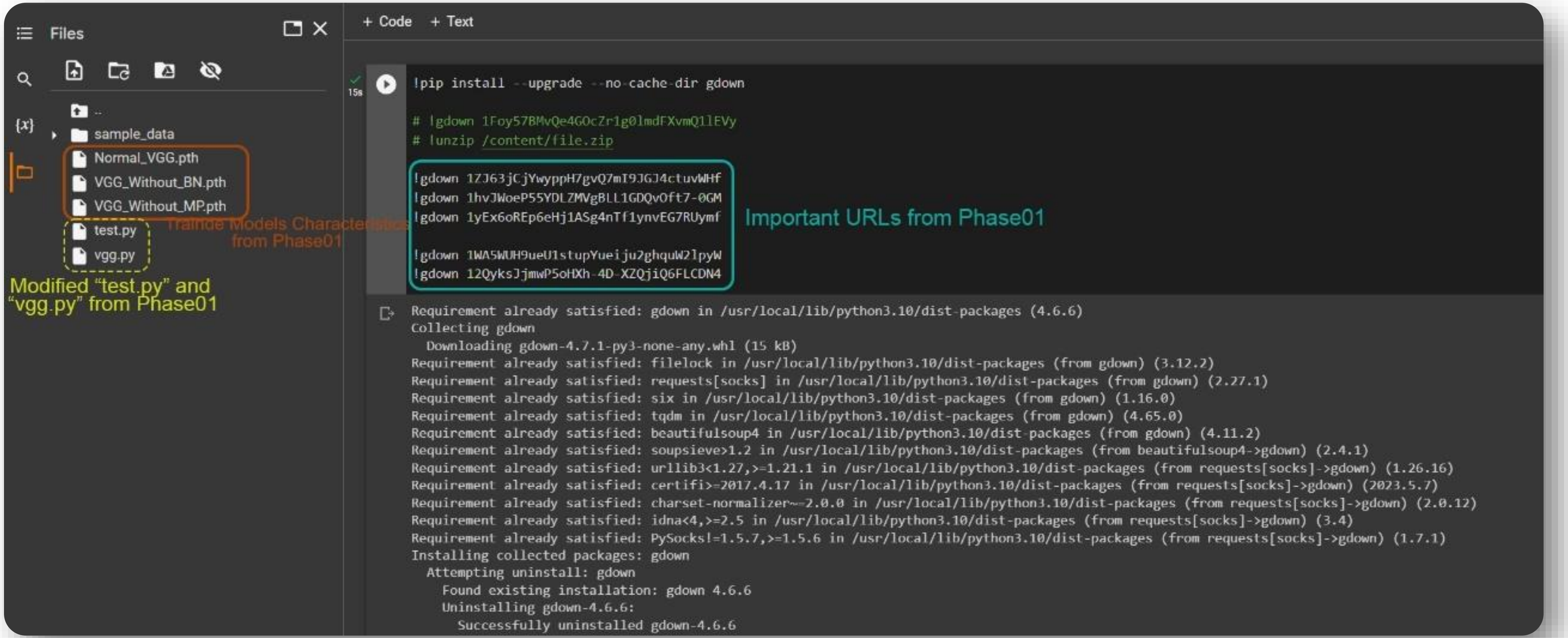
- CIFAR10 Dataset

✓ The CIFAR-10 dataset is a widely used **benchmark** dataset in the field of computer vision and machine learning. CIFAR-10 consists of **60,000 color images**, each measuring **32x32 pixels**, divided into **10 different classes**. Each class contains 6,000 images.



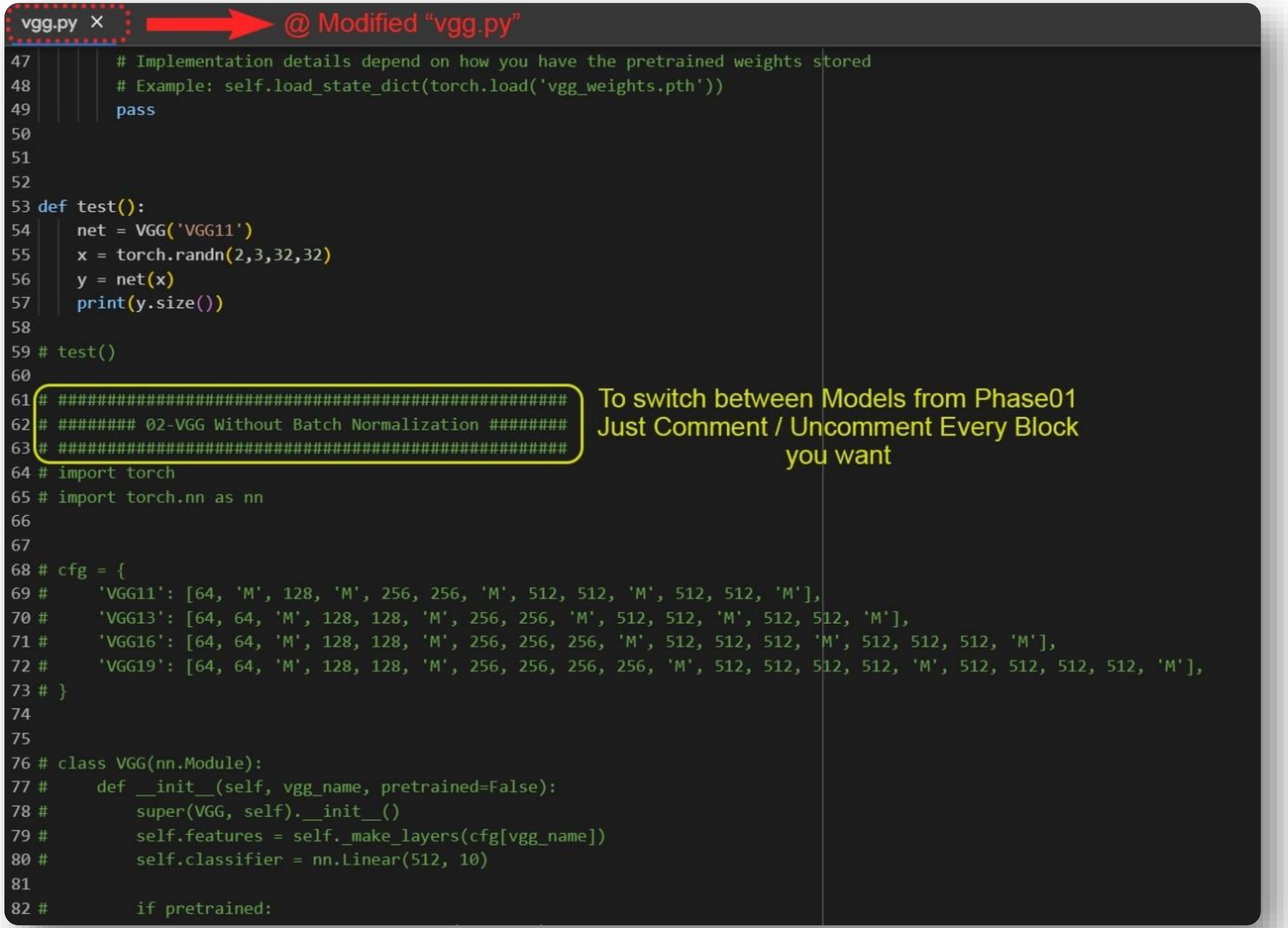
Introduction

- Code Requirements



Introduction

- Code Requirements
  - ✓ At modified files called “vgg.py” and “test.py”, we will switch between three various Models from phase 01.




```python
            # Implementation details depend on how you have the pretrained weights stored
            # Example: self.load_state_dict(torch.load('vgg_weights.pth'))
            pass



def test():
    net = VGG('VGG11')
    x = torch.randn(2,3,32,32)
    y = net(x)
    print(y.size())

# test()

# ########################################################
# ######## 02-VGG Without Batch Normalization ########
# ########################################################
# import torch
# import torch.nn as nn


# cfg = {
#     'VGG11': [64, 'M', 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
#     'VGG13': [64, 64, 'M', 128, 128, 'M', 256, 256, 'M', 512, 512, 'M', 512, 512, 'M'],
#     'VGG16': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 'M', 512, 512, 512, 'M', 512, 512, 512, 'M'],
#     'VGG19': [64, 64, 'M', 128, 128, 'M', 256, 256, 256, 256, 'M', 512, 512, 512, 512, 'M', 512, 512, 512, 512, 'M'],
# }


# class VGG(nn.Module):
#     def __init__(self, vgg_name, pretrained=False):
#         super(VGG, self).__init__()
#         self.features = self._make_layers(cfg[vgg_name])
#         self.classifier = nn.Linear(512, 10)

#         if pretrained:
```

Introduction

- Code Requirements
    - ✓ At modified files called "vgg.py" and "test.py", we will switch between three various Models from phase 01.

test.py ✕ @ Modified "test.py"

To switch between Models from Phase01
Commnet intended Line

```python
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=shuffle)


    return test_loader

def cifar10_test(device, model_name: str = 'ResNet18',batch_size = batch_size, n_examples = n_examples,checkpoint_type: str='p
    batch_size = batch_size
    os.makedirs(os.path.join('data', 'torchvision'), exist_ok=True)
    os.makedirs(os.path.join('results', 'cifar10'), exist_ok=True)

    n_examples = n_examples
    images, labels = load_cifar10(n_examples=n_examples, data_dir='data/torchvision')
    test_dataset = torch.utils.data.TensorDataset(images, labels)
    test_loader = torch.utils.data.DataLoader(test_dataset, batch_size=batch_size, shuffle=False)

    if  checkpoint_type == 'pth':
        if model_name == 'vgg' :
            model = VGG(vgg_name='VGG16', pretrained=True)
            model = nn.DataParallel(model)
            checkpoint = torch.load(r"/content/ML_Graduate_Project_TA/Phase_1/models/state_dict/Normal_VGG.pth") #01
            # checkpoint = torch.load(r"/content/ML_Graduate_Project_TA/Phase_1/models/state_dict/VGG_Without_BN.pth") #02
            # checkpoint = torch.load(r"/content/ML_Graduate_Project_TA/Phase_1/models/state_dict/VGG_Without_MP.pth") #03

            model.load_state_dict(checkpoint['net'])
            model = normalize_model(model, mean = (.4914, 0.4822, 0.4465), std = (0.2023, 0.1994, 0.2010))
        elif model_name == 'vgg' :
            model = VGG(vgg_name='VGG16', pretrained=True)
            model = normalize_model(model, mean = (.4914, 0.4822, 0.4465), std = (0.2471, 0.2435, 0.2616))

        else:
            raise ValueError(f"Model {model_name} not found")

    elif  checkpoint_type == 'pt':
        if model_name == 'ResNet18':
            model = ResNet18()
            model = nn.DataParallel(model) # note that please check the repo, if it uses the Dataprallel use this line!
            model.load_state_dict(torch.load(r"ckpth.pt"))
            model = normalize_model(model, mean = (.4914, 0.4822, 0.4465), std = (0.2023, 0.1994, 0.2010))
        elif model_name == 'vgg':
            model = VGG(vgg_name='VGG16', pretrained=True)
            model = nn.DataParallel(model) # note that please check the repo, if it uses the Dataprallel use this line!
            model.load_state_dict(torch.load(r"ckpth.pt"))
            model = normalize_model(model, mean = (.4914, 0.4822, 0.4465), std = (0.2471, 0.2435, 0.2616))

```

Introduction

- ## Code Requirements

  ✓ Installing DDN Repository Packages:

```
pip install git+https://github.com/jeromerony/fast_adversarial

Collecting git+https://github.com/jeromerony/fast_adversarial
  Cloning https://github.com/jeromerony/fast_adversarial to /tmp/pip-req-build-l7_rbkht
  Running command git clone --filter=blob:none --quiet https://github.com/jeromerony/fast_adversarial /tmp/pip-req-build-l7_rbkht
  Resolved https://github.com/jeromerony/fast_adversarial to commit 45210b7c79e2deaeac9845d6c901dc2580d6e316
  Preparing metadata (setup.py) ... done
Requirement already satisfied: torch>=0.4.1 in /usr/local/lib/python3.10/dist-packages (from fast-adversarial==0.1) (2.0.1+cu118)
Requirement already satisfied: torchvision>=0.2.1 in /usr/local/lib/python3.10/dist-packages (from fast-adversarial==0.1) (0.15.2+cu118)
Requirement already satisfied: tqdm>=4.23.4 in /usr/local/lib/python3.10/dist-packages (from fast-adversarial==0.1) (4.65.0)
Collecting visdom>=0.1.8 (from fast-adversarial==0.1)
  Downloading visdom-0.2.4.tar.gz (1.4 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.4/1.4 MB 38.2 MB/s eta 0:00:00
  Preparing metadata (setup.py) ... done
Collecting foolbox>=1.7.0 (from fast-adversarial==0.1)
  Downloading foolbox-3.3.3-py3-none-any.whl (1.7 MB)
     ━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━━ 1.7/1.7 MB 70.4 MB/s eta 0:00:00
```

By this part of code, the required packages will be installed

Introduction

- Code Requirements

```python
import argparse
import torch
import time
from torch.utils import data
from torchvision import datasets, transforms
from torchvision.utils import save_image, make_grid
import warnings
warnings.filterwarnings('ignore')
from vgg import VGG
from fast_adv.attacks import DDN, CarliniWagnerL2
from fast_adv.utils import requires_grad_, l2_norm
import matplotlib.pyplot as plt
import os
%matplotlib inline

torch.manual_seed(42)
# device = torch.device('cuda:0' if torch.cuda.is_available() else 'cpu')
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
data_path = 'data/cifar10'  # Change this if you already downloaded CIFAR-10 elsewhere
model_path = '/content/VGG_Without_MP.pth'  # Specify the path to your model file
```

1. Adding required libraries
2. Definig Model path

```python
# Loading the data
dataset = datasets.CIFAR10(data_path, train=False,
                           transform=transforms.ToTensor(),
                           download=True)
loader = data.DataLoader(dataset, shuffle=False, batch_size=16)

x, y = next(iter(loader))
x = x.to(device)
y = y.to(device)

x_cpu = x.cpu()
grid_image = make_grid(x_cpu, nrow=16).permute(1, 2, 0)
plt.imshow(grid_image)
plt.axis('off')
plt.show()
```

Loading CIFAR10

```
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to data/cifar10/cifar-10-python.tar.gz
100%|██████████| 170498071/170498071 [00:13<00:00, 12625167.17it/s]
Extracting data/cifar10/cifar-10-python.tar.gz to data/cifar10
```

Introduction

- Code Requirements
  - ✓ Applying DDN 100 Attack to the model.

```python
import torch
import torch.nn as nn

print('Loading model')

if not os.path.exists(model_path):
    import urllib
    print('Downloading model')
    urllib.request.urlretrieve(model_url, model_path)


model = VGG(vgg_name='VGG16', pretrained=True)
model = nn.DataParallel(model)
checkpoint = torch.load(r"/content/VGG_Without_MP.pth")
# model.load_state_dict(torch.load(model_path))
model.load_state_dict(checkpoint['net'])
model.eval().to(device)
requires_grad_(model, False)
```

Loading Intended Model

```
Loading model
```

```python
print('Running DDN 100 attack')
attacker = DDN(steps=100, device=device)
start = time.time()
ddn_atk = attacker.attack(model, x, labels=y, targeted=False)
ddn_time = time.time() - start
print('Completed in {:.2f}s'.format(ddn_time))

plt.imshow(make_grid(ddn_atk.cpu(), nrow=16).permute(1, 2, 0))
plt.axis('off');
```

Running DDN attack

```
Running DDN 100 attack
Completed in 13.90s
```

Elapsed Time

Introduction

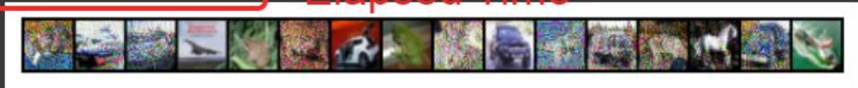
- Code Requirements
  - ✓ Applying C&W 4x25 Attack to the model.

```python
print('Running C&W 4 x 25 attack (limited to 100 iterations)')
cwattacker100 = CarliniWagnerL2(device=device,
                               image_constraints=(0, 1),
                               num_classes=10,
                               search_steps=4,
                               max_iterations=25,
                               learning_rate=0.5,
                               initial_const=1.0)

start = time.time()
cw100_atk = cwattacker100.attack(model, x, labels=y, targeted=False)
cw100_time = time.time() - start
print('Completed in {:.2f}s'.format(cw100_time))

plt.imshow(make_grid(cw100_atk.cpu(), nrow=16).permute(1,2,0))
plt.axis('off');
```

Running C&W 4x25 Attack

```
Running C&W 4 x 25 attack (limited to 100 iterations)
Completed in 0.90s
```

Elapsed Time



Introduction

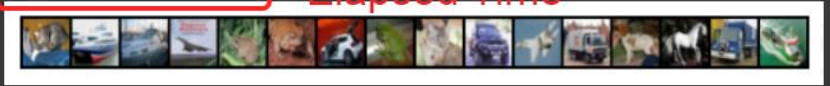
- Code Requirements
  - ✓ Applying C&W 9x10000 Attack to the model.

```
print('Running C&W 9 x 10000 attack')
cwattacker = CarliniWagnerL2(device=device,
                             image_constraints=(0, 1),
                             num_classes=10)

start = time.time()
cw_atk = cwattacker.attack(model, x, labels=y, targeted=False)
cw_time = time.time() - start
print('Completed in {:.2f}s'.format(cw_time))

plt.imshow(make_grid(cw_atk.cpu(), nrow=16).permute(1,2,0))
plt.axis('off');
```

Running C&W 9x10000 Attack

```
Running C&W 9 x 10000 attack
Completed in 1780.00s
```

Elapsed Time



Amirkabir University of Technology
Faculty of Electrical Engineering

ML – Final Project

Chapter2: Normal VGG

The process of Applying DDN100 Attack to Normal VGG and recording the results.

Normal VGG

- Results

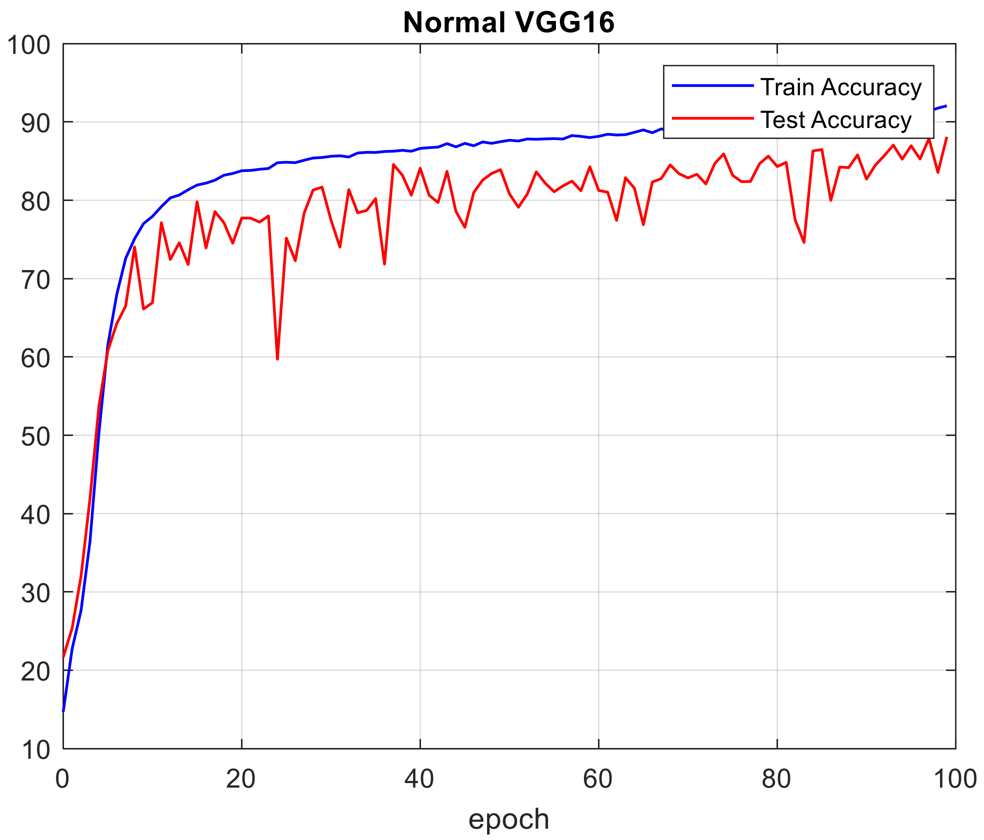
  ✓ After applying the attacks to normal VGG:

```
all_imgs = torch.cat((x, cw100_atk, cw_atk, ddn_atk))
img_grid = make_grid(all_imgs, nrow=16, pad_value=0)
plt.imshow(img_grid.cpu().permute(1, 2, 0))
plt.axis('off')

# Print metrics
pred_orig = model(x).argmax(dim=1).cpu()
pred_cw = model(cw_atk).argmax(dim=1).cpu()
pred_cw100 = model(cw100_atk).argmax(dim=1).cpu()
pred_ddn = model(ddn_atk).argmax(dim=1).cpu()
print('C&W 4 x 25 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
cw100_time,
(pred_cw100 != y.cpu()).float().mean().item() * 100,
l2_norm(cw100_atk - x).mean().item(),
l2_norm(cw100_atk - x).median().item()
))
print('C&W 9 x 10000 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
cw_time,
(pred_cw != y.cpu()).float().mean().item() * 100,
l2_norm(cw_atk - x).mean().item(),
l2_norm(cw_atk - x).median().item()
))
print('DDN 100 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
ddn_time,
(pred_ddn != y.cpu()).float().mean().item() * 100,
l2_norm(ddn_atk - x).mean().item(),
l2_norm(ddn_atk - x).median().item()
))
print()
print('Figure: top row: original images; 2nd: C&W 4x25 atk; 3rd: C&W 9x10000 atk; 4th: DDN 100 atk')
```

```
C&W 4 x 25 done in 0.2s: Success: 100.00%, Mean L2: 8.8299, Median L2: 3.7398.
C&W 9 x 10000 done in 228.7s: Success: 100.00%, Mean L2: 0.2423, Median L2: 0.2277.
DDN 100 done in 7.3s: Success: 100.00%, Mean L2: 0.2384, Median L2: 0.2368.

Figure: top row: original images; 2nd: C&W 4x25 atk; 3rd: C&W 9x10000 atk; 4th: DDN 100 atk
```

Normal VGG

- Results
  - ✓ From Phase 01:



Normal VGG

- Results

  ✓ Overall:

Model Name	Clean Acc. (Train)	Clean Acc. (Test)	L_2 $Mean$	L_2 $Median$
Normal VGG	**92%**	**88%**	**0.2384**	**0.2368**

ML – Final Project

Chapter3: VGG Without B. N.

The process of Applying DDN100 Attack to VGG Without B. N. and recording the results.

VGG Without B. N.

- Results

  ✓ After applying the attacks to VGG Without B. N.:

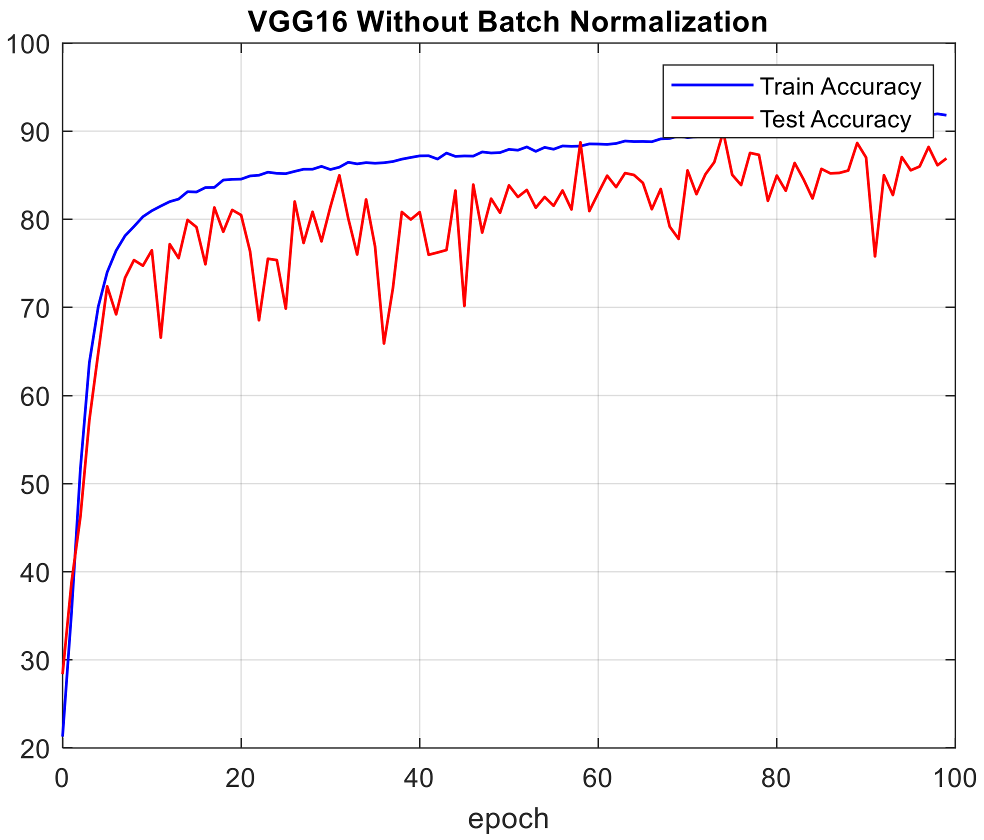
```python
all_imgs = torch.cat((x, cw100_atk, cw_atk, ddn_atk))
img_grid = make_grid(all_imgs, nrow=16, pad_value=0)
plt.imshow(img_grid.cpu().permute(1, 2, 0))
plt.axis('off')

# Print metrics
pred_orig = model(x).argmax(dim=1).cpu()
pred_cw = model(cw_atk).argmax(dim=1).cpu()
pred_cw100 = model(cw100_atk).argmax(dim=1).cpu()
pred_ddn = model(ddn_atk).argmax(dim=1).cpu()
print('C&W 4 x 25 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
cw100_time,
(pred_cw100 != y.cpu()).float().mean().item() * 100,
l2_norm(cw100_atk - x).mean().item(),
l2_norm(cw100_atk - x).median().item()
))
print('C&W 9 x 10000 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
cw_time,
(pred_cw != y.cpu()).float().mean().item() * 100,
l2_norm(cw_atk - x).mean().item(),
l2_norm(cw_atk - x).median().item()
))
print('DDN 100 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
ddn_time,
(pred_ddn != y.cpu()).float().mean().item() * 100,
l2_norm(ddn_atk - x).mean().item(),
l2_norm(ddn_atk - x).median().item()
))
print()
print('Figure: top row: original images; 2nd: C&W 4x25 atk; 3rd: C&W 9x10000 atk; 4th: DDN 100 atk')
```

```
C&W 4 x 25 done in 0.2s: Success: 100.00%, Mean L2: 3.2841, Median L2: 0.0000.
C&W 9 x 10000 done in 262.8s: Success: 100.00%, Mean L2: 0.0756, Median L2: 0.0000.
DDN 100 done in 7.7s: Success: 100.00%, Mean L2: 0.0675, Median L2: 0.0319.

Figure: top row: original images; 2nd: C&W 4x25 atk; 3rd: C&W 9x10000 atk; 4th: DDN 100 atk
```

VGG Without B. N.

- Results

  ✓ From Phase01:



VGG Without B. N.

- Results
  - ✓ Overall:

Model Name	Clean Acc. (Train)	Clean Acc. (Test)	L_2 *Mean*	L_2 *Median*
VGG Without Batch Normalization	**91%**	**88%**	**0.0675**	**0.0319**

Amirkabir University of Technology
Faculty of Electrical Engineering

ML – Final Project

Chapter4: VGG Without M. P.

The process of Applying DDN100 Attack to VGG Without M. P. and recording the results.

Machine Learning
Alireza Ansari

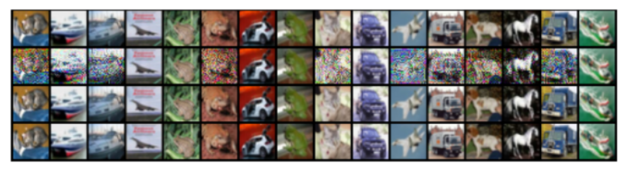
July 14 . 2023

Normal VGG

- Results
  - ✓ After applying the attacks to VGG Without M. P.:

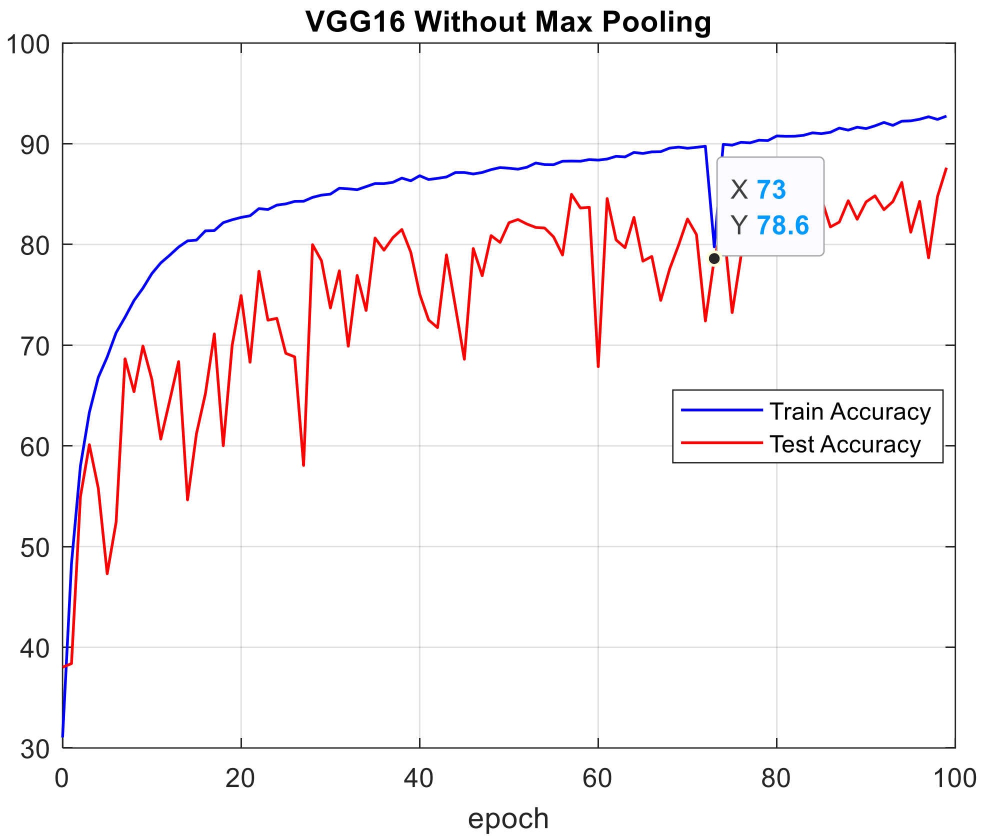
```
all_imgs = torch.cat((x, cw100_atk, cw_atk, ddn_atk))
img_grid = make_grid(all_imgs, nrow=16, pad_value=0)
plt.imshow(img_grid.cpu().permute(1, 2, 0))
plt.axis('off')

# Print metrics
pred_orig = model(x).argmax(dim=1).cpu()
pred_cw = model(cw_atk).argmax(dim=1).cpu()
pred_cw100 = model(cw100_atk).argmax(dim=1).cpu()
pred_ddn = model(ddn_atk).argmax(dim=1).cpu()
print('C&W 4 x 25 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
cw100_time,
(pred_cw100 != y.cpu()).float().mean().item() * 100,
l2_norm(cw100_atk - x).mean().item(),
l2_norm(cw100_atk - x).median().item()
))
print('C&W 9 x 10000 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
cw_time,
(pred_cw != y.cpu()).float().mean().item() * 100,
l2_norm(cw_atk - x).mean().item(),
l2_norm(cw_atk - x).median().item()
))
print('DDN 100 done in {:.1f}s: Success: {:.2f}%, Mean L2: {:.4f}, Median L2: {:.4f}.'.format(
ddn_time,
(pred_ddn != y.cpu()).float().mean().item() * 100,
l2_norm(ddn_atk - x).mean().item(),
l2_norm(ddn_atk - x).median().item()
))
print()
print('Figure: top row: original images; 2nd: C&W 4x25 atk; 3rd: C&W 9x10000 atk; 4th: DDN 100 atk')
```

```
C&W 4 x 25 done in 0.9s: Success: 100.00%, Mean L2: 6.6092, Median L2: 8.8581.
C&W 9 x 10000 done in 1780.0s: Success: 100.00%, Mean L2: 0.0928, Median L2: 0.0440.
DDN 100 done in 14.1s: Success: 100.00%, Mean L2: 0.0784, Median L2: 0.0479.

Figure: top row: original images; 2nd: C&W 4x25 atk; 3rd: C&W 9x10000 atk; 4th: DDN 100 atk
```



VGG Without M. P.

- Results
  - ✓ From Phase01:



VGG Without M. P.

- Results

  ✓ Overall:

Model Name	Clean Acc. (Train)	Clean Acc. (Test)	L_2 *Mean*	L_2 *Median*
VGG Without Batch Normalization	**92%**	**87%**	**0.0784**	**0.0479**

Amirkabir University of Technology
Faculty of Electrical Engineering

ML – Final Project

Chapter5: Conclusion

Comparing the results.

Conclusion

- Ambiguities

  ✓ It can be seen that after removing all the pooling layers, the accuracy of the model being trained is always upward and reaches 100% in the final epochs, which can indicate overfitting.

  ✓ Also, the lack of validation data and evaluation of the model by it also makes it difficult to judge the relationship of the model’s conditions.

ML – Final Project

Chapter6: References

Introduce The References used in This Presentation.

References

[1] C. M., *Pattern Recognition and Machine Learning*, 1st ed. New York, NY: Springer, 2006.

[2] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. Nashville, TN: John Wiley & Sons, 2000.

[3] M. Elgendy, *Deep learning for vision systems*. New York, NY: Manning Publications, 2021.


Thanks for Your Attention