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).

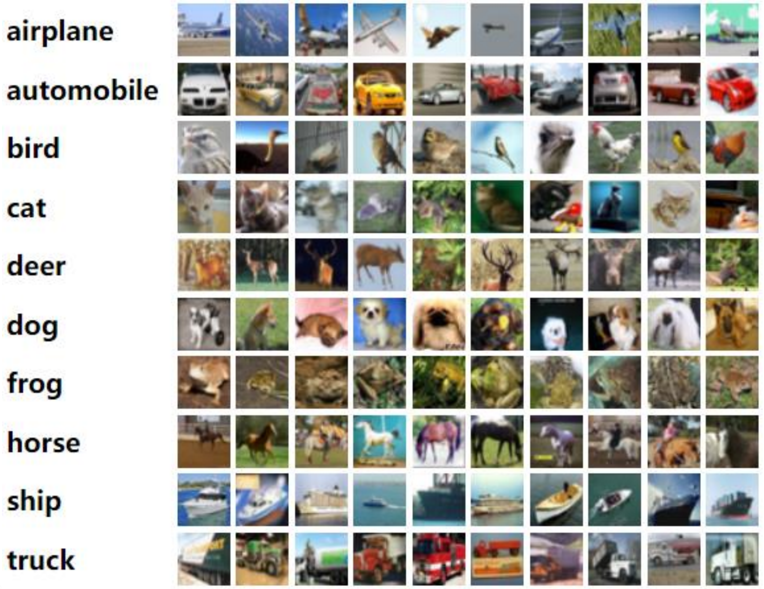
  ✓ The building components are exactly the same as those in **LeNet** and **AlexNet**, except that VGGNet is an even deeper network with more **convolutional**, **pooling**, and **dense layers**.

  ✓ 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

- Batch Normalization

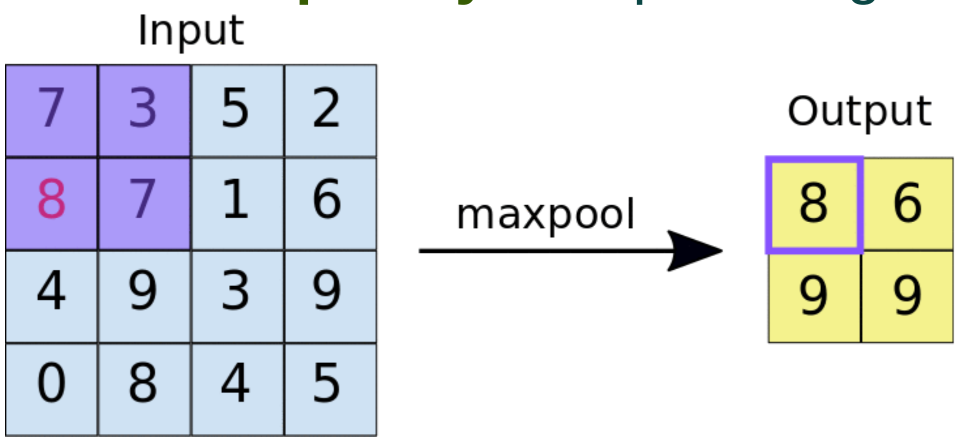
  ✓ Batch normalization is a technique used in neural networks to **normalize the inputs of each layer**. It helps address the issue of internal covariate shift, which is the change in the distribution of network activations as the parameters of the preceding layers change during training.

  ✓ Usually inserted **after Fully Connected or Convolutional Layers**, and **before Nonlinearity**.

Introduction

- Max Pooling Layers

  ✓ The max pooling layer is a component of neural networks used for down-sampling or **reducing the spatial dimensions** of the input data. It divides the input into non-overlapping regions and takes the maximum value within each region, discarding the rest. This operation helps to **extract the most important features while reducing the computational complexity** and providing a form of translation invariance.



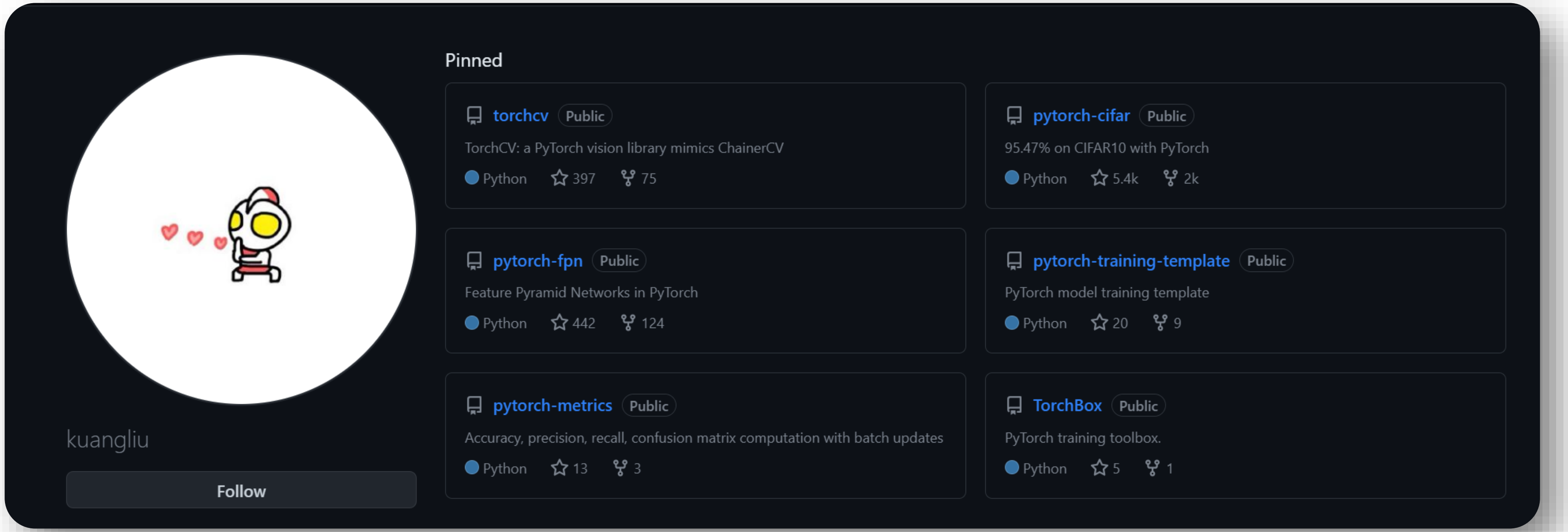
ML – Final Project

Chapter2: Normal VGG

The process of Training Normal VGG and recording the results.

Normal VGG

- Reference!
  - ✓ To Train and Test the VGG16 in Normal mode, we use **Kunagliu** Repository.



Normal VGG

- Before Running the Code

  ✓ @ main.py, Define our desired Network (VGG16).

```python
# Model
print('==> Building model..')
net = VGG('VGG16')
# net = ResNet18()
# net = PreActResNet18()
# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
# net = SimpleDLA()
```

```python
136     # Save checkpoint.
137     acc = 100.*correct/total
138     if acc > best_acc:
139         print('Saving..')
140         state = {
141             'net': net.state_dict(),
142             'acc': acc,
143             'epoch': epoch,
144         }
145         if not os.path.isdir('checkpoint'):
146             os.mkdir('checkpoint')
147         torch.save(state, './checkpoint/ckpt.pth')
148         best_acc = acc
```

  ✓ Create a “checkpoint” folder and address it to line 147 for saving the results in “.pth” format after 200 epochs.

Normal VGG

- Code Structure

Cloning Kuangliu pytorch_cifar Repository

Command to Train Normal VGG 16 for 200 epochs

```
!git clone https://github.com/kuangliu/pytorch-cifar.git
```

```
Cloning into 'pytorch-cifar'...
remote: Enumerating objects: 382, done.
remote: Counting objects: 100% (382/382), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 382 (delta 209), reused 355 (delta 197), pack-reused 0
Receiving objects: 100% (382/382), 77.42 KiB | 5.53 MiB/s, done.
Resolving deltas: 100% (209/209), done.
```

```
%cd /content/pytorch-cifar
```

```
/content/pytorch-cifar
```

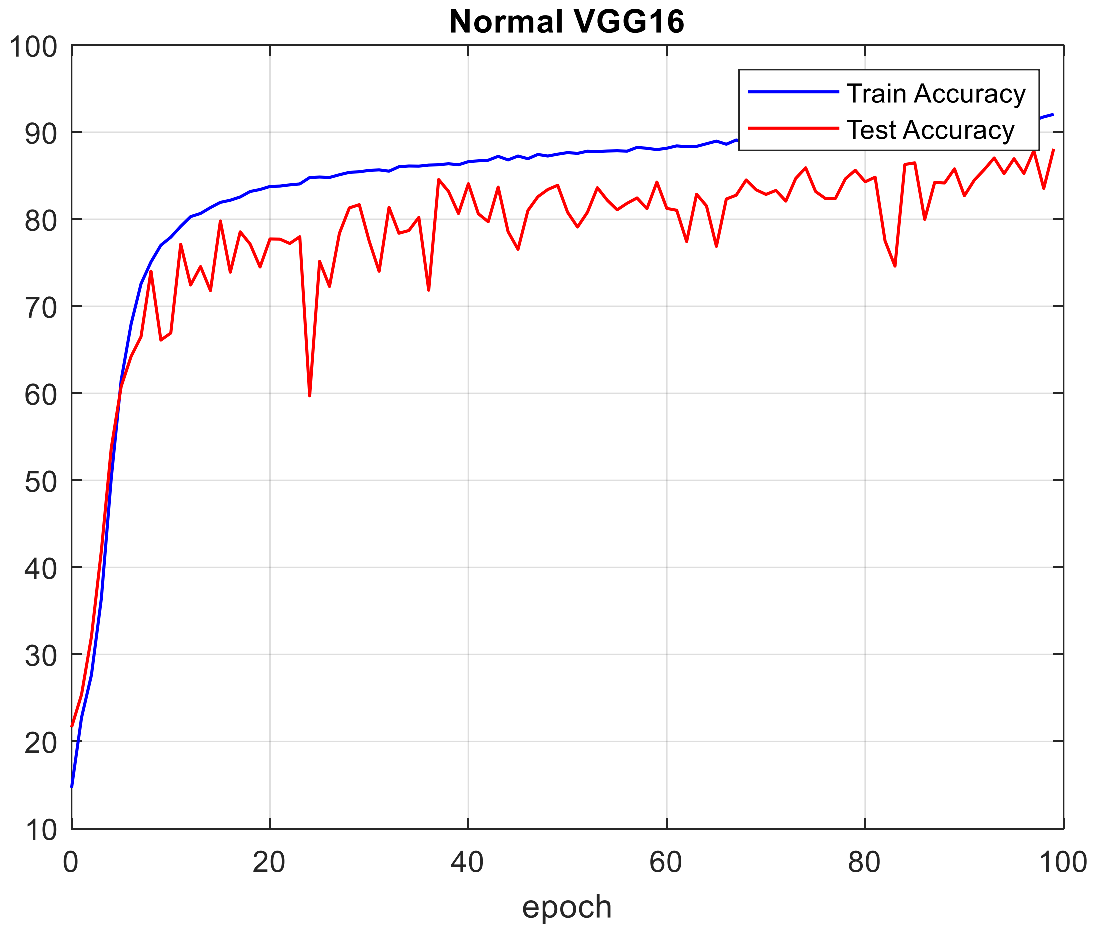
```
# Start training with:
!python main.py

# You can manually resume the training with:
!python main.py --resume --lr=0.01
```

```
==> Preparing data..
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100% 170498071/170498071 [00:03<00:00, 52056434.58it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
==> Building model..

Epoch: 0
```

Normal VGG

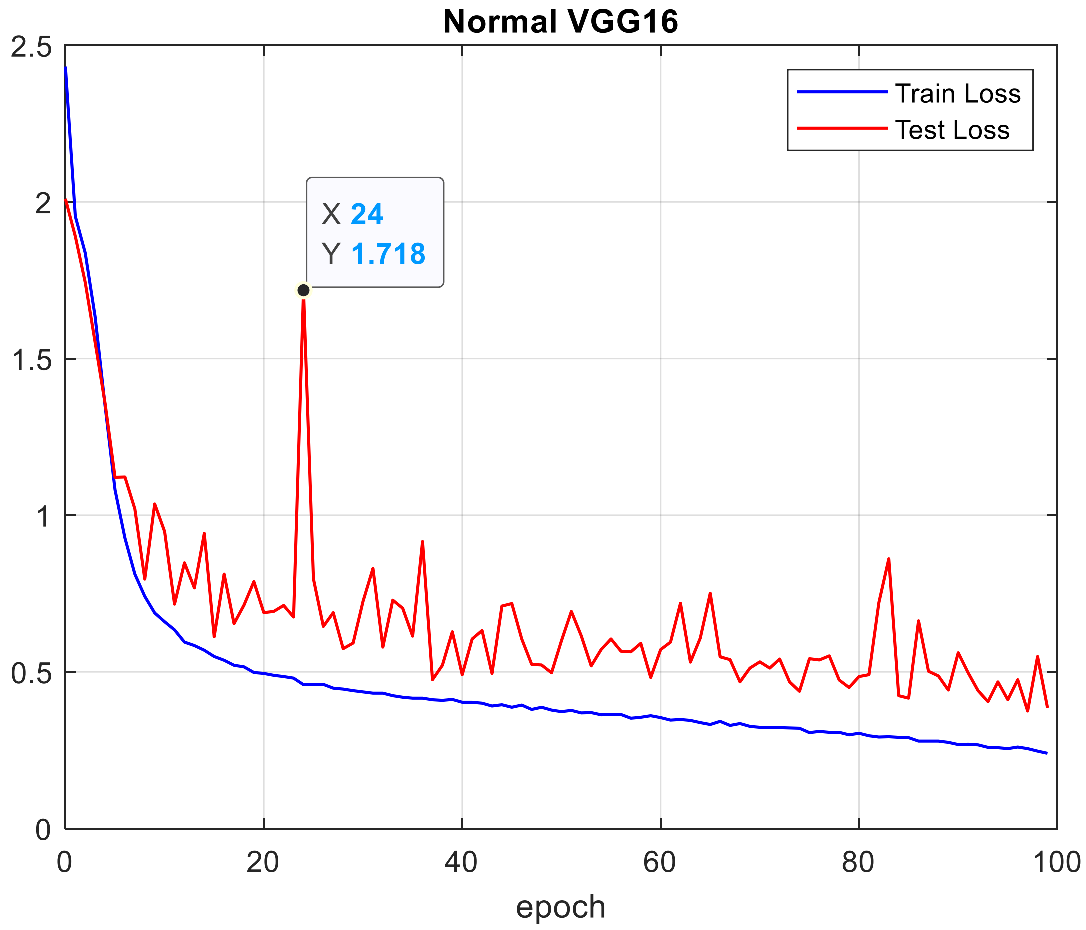
- Output

  ✓ After the Iteration of 200 epochs, the results are obtained as follows:



Normal VGG

- Output
  - ✓ After the Iteration of 200 epochs, the results are obtained as follows:



Normal VGG

- Screenshot of Iterating all epochs

```
Epoch: 196
 [================================================================>]  Step: 38ms | Tot: 25s343ms | Loss: 0.001 | Acc: 99.978% (49989/50000) 391/391
 [================================================================>]  Step: 21ms | Tot: 2s899ms | Loss: 0.290 | Acc: 93.850% (9385/10000) 100/100

Epoch: 197
 [================================================================>]  Step: 39ms | Tot: 25s561ms | Loss: 0.002 | Acc: 99.970% (49985/50000) 391/391
 [================================================================>]  Step: 22ms | Tot: 2s744ms | Loss: 0.289 | Acc: 93.820% (9382/10000) 100/100

Epoch: 198
 [================================================================>]  Step: 37ms | Tot: 25s829ms | Loss: 0.001 | Acc: 99.978% (49989/50000) 391/391
 [================================================================>]  Step: 25ms | Tot: 2s863ms | Loss: 0.290 | Acc: 93.840% (9384/10000) 100/100

Epoch: 199
 [================================================================>]  Step: 38ms | Tot: 27s756ms | Loss: 0.002 | Acc: 99.974% (49987/50000) 391/391
 [================================================================>]  Step: 31ms | Tot: 2s954ms | Loss: 0.289 | Acc: 93.830% (9383/10000) 100/100
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..
==> Resuming from checkpoint..
```

Amirkabir University of Technology
Faculty of Electrical Engineering

ML – Final Project

Chapter3: VGG Without B. N.

Training and Testing the VGG Without Batch Normalization.

VGG Without B. N.

- Before run the code

  ✓ @ main.py, Define our desired Network (VGG16).

```python
55    # Model
56    print('==> Building model..')
57    net = VGG('VGG16')
58    # net = ResNet18()
59    # net = PreActResNet18()
60    # net = GoogLeNet()
61    # net = DenseNet121()
62    # net = ResNeXt29_2x64d()
63    # net = MobileNet()
64    # net = MobileNetV2()
65    # net = DPN92()
66    # net = ShuffleNetG2()
67    # net = SENet18()
68    # net = ShuffleNetV2(1)
69    # net = EfficientNetB0()
70    # net = RegNetX_200MF()
71    # net = SimpleDLA()
```

```python
136        # Save checkpoint.
137        acc = 100.*correct/total
138        if acc > best_acc:
139            print('Saving..')
140            state = {
141                'net': net.state_dict(),
142                'acc': acc,
143                'epoch': epoch,
144            }
145            if not os.path.isdir('checkpoint'):
146                os.mkdir('checkpoint')
147            torch.save(state, './checkpoint/ckpt.pth')
148            best_acc = acc
```

✓ Create a “checkpoint” folder and address it to line 147 for saving the results in “.pth” format after 200 epochs.

VGG Without B. N.

- Modifying the VGG Model

  ✓ @ "vgg.py", By modifying this part of the code, we removed the B. N. from last Conv. Block.

```python
def _make_layers(self, cfg):
    layers = []
    in_channels = 3
    for x in cfg:
        if x == 'M':
            layers += [nn.MaxPool2d(kernel_size=2, stride=2)]
        else:
            layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                       nn.BatchNorm2d(x),
                       nn.ReLU(inplace=True)]
            in_channels = x

    # Remove the batch normalization layer from the last convolutional block
    last_block_index = len(layers) - 3
    layers = layers[:last_block_index] + layers[last_block_index + 2:]

    layers += [nn.AvgPool2d(kernel_size=1, stride=1)]
    return nn.Sequential(*layers)
```

VGG Without B. N.

- Code Structure

Cloning Kuangliu pytorch_cifar Repository And applying previous slide’s modification

```
!git clone https://github.com/kuangliu/pytorch-cifar.git
```

```
Cloning into 'pytorch-cifar'...
remote: Enumerating objects: 382, done.
remote: Counting objects: 100% (382/382), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 382 (delta 209), reused 355 (delta 197), pack-reused 0
Receiving objects: 100% (382/382), 77.42 KiB | 5.53 MiB/s, done.
Resolving deltas: 100% (209/209), done.
```

```
%cd /content/pytorch-cifar
```

```
/content/pytorch-cifar
```

Command to Train VGG 16 Without B. N. for 200 epochs

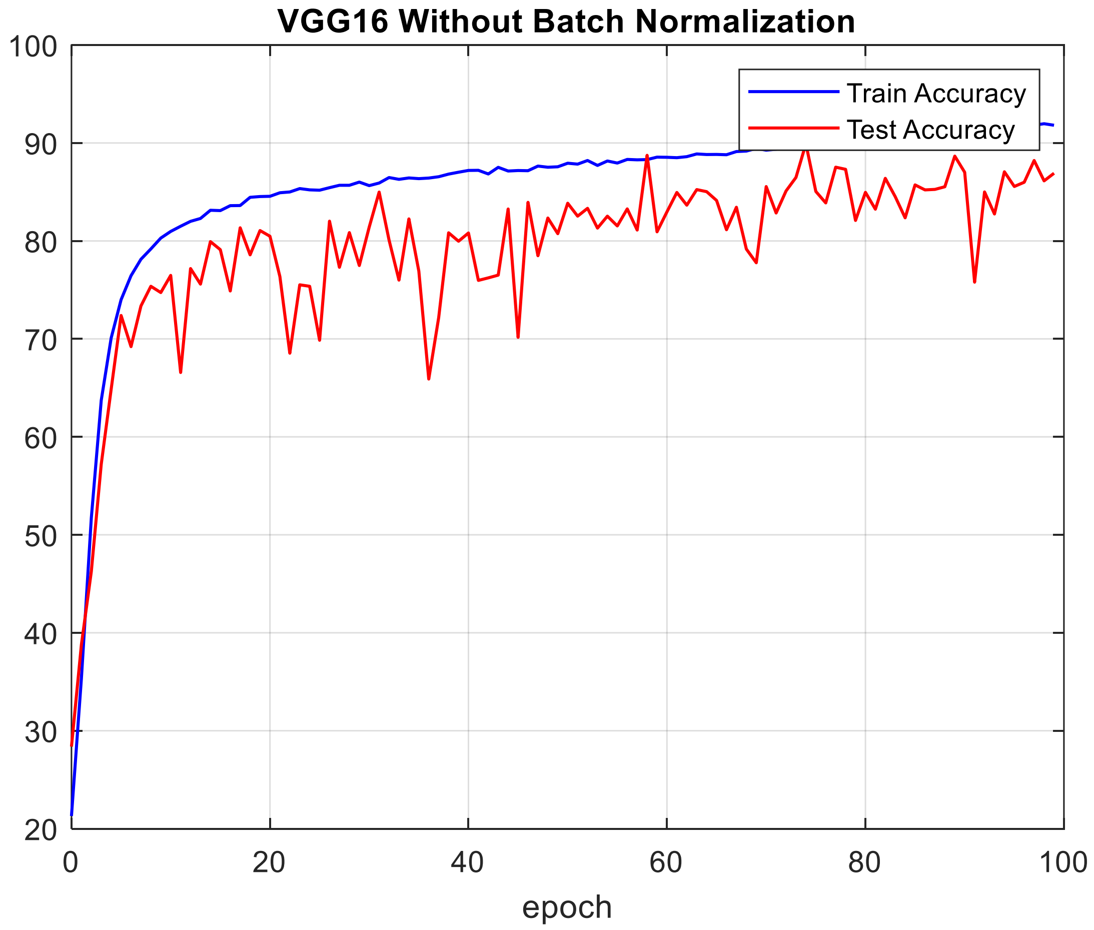
```
# Start training with:
!python main.py

# You can manually resume the training with:
!python main.py --resume --lr=0.01
```

```
==> Preparing data..
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100% 170498071/170498071 [00:03<00:00, 52056434.58it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
==> Building model..

Epoch: 0
```

VGG Without B. N.

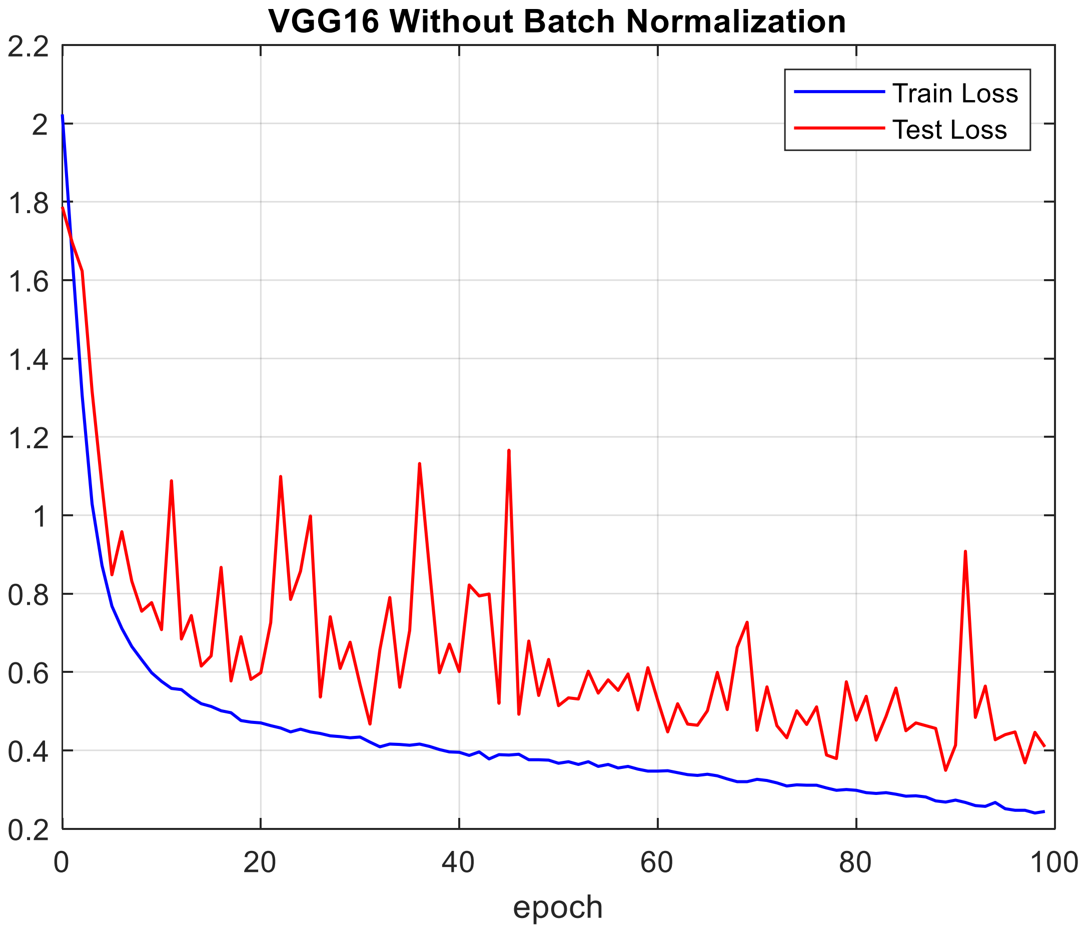
- Output

  ✓ After the Iteration of 200 epochs, the results are obtained as follows:



VGG Without B. N.

- Output
  - ✓ After the Iteration of 200 epochs, the results are obtained as follows:



VGG Without B. N.

- Screenshot of Iterating all epochs

```
Epoch: 195
 [=================================================================>]  Step: 38ms | Tot: 28s672ms | Loss: 0.001 | Acc: 99.982% (49991/50000) 391/391
 [=================================================================>]  Step: 24ms | Tot: 3s126ms | Loss: 0.263 | Acc: 94.150% (9415/10000) 100/100
Saving..

Epoch: 196
 [=================================================================>]  Step: 40ms | Tot: 28s482ms | Loss: 0.001 | Acc: 99.986% (49993/50000) 391/391
 [=================================================================>]  Step: 25ms | Tot: 3s313ms | Loss: 0.265 | Acc: 94.100% (9410/10000) 100/100

Epoch: 197
 [=================================================================>]  Step: 41ms | Tot: 29s118ms | Loss: 0.001 | Acc: 99.982% (49991/50000) 391/391
 [=================================================================>]  Step: 15ms | Tot: 3s181ms | Loss: 0.263 | Acc: 94.180% (9418/10000) 100/100
Saving..

Epoch: 198
 [=================================================================>]  Step: 40ms | Tot: 28s584ms | Loss: 0.001 | Acc: 99.982% (49991/50000) 391/391
 [=================================================================>]  Step: 28ms | Tot: 3s244ms | Loss: 0.264 | Acc: 94.080% (9408/10000) 100/100

Epoch: 199
 [=================================================================>]  Step: 39ms | Tot: 29s140ms | Loss: 0.001 | Acc: 99.974% (49987/50000) 391/391
 [=================================================================>]  Step: 26ms | Tot: 3s848ms | Loss: 0.266 | Acc: 94.030% (9403/10000) 100/100
torch.Size([2, 10])
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..
==> Resuming from checkpoint..
```

ML – Final Project

Chapter4: VGG Without M. P.

Training and Testing the VGG In the absence of Max Pooling Layers.

VGG Without M. P.

- Before run the code
  - ✓ @ main.py, Define our desired Network (VGG16).

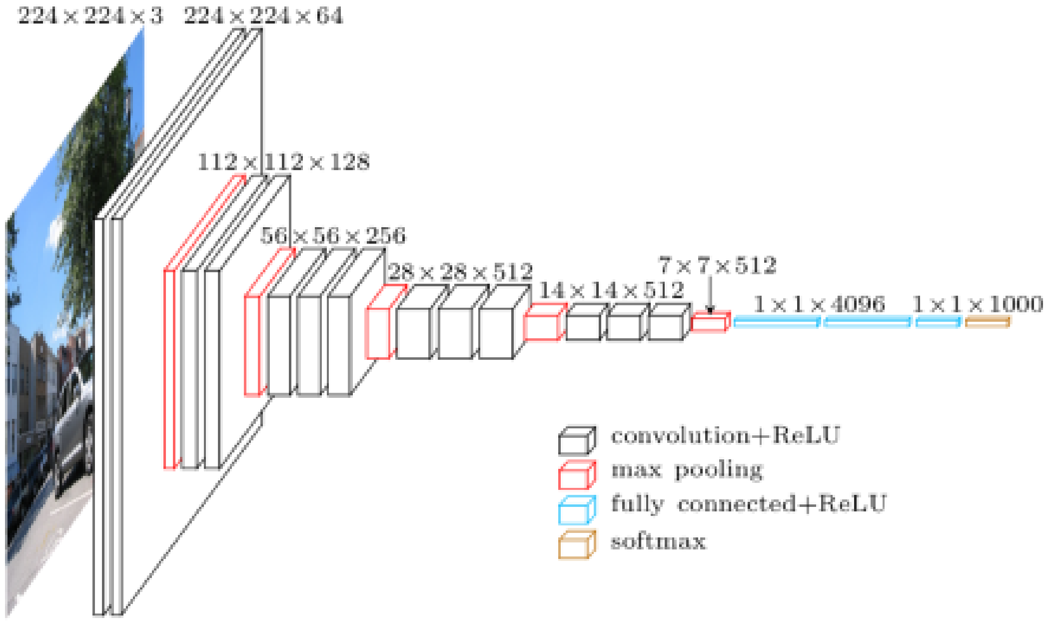
```python
# Model
print('==> Building model..')
net = VGG('VGG16')
# net = ResNet18()
# net = PreActResNet18()
# net = GoogLeNet()
# net = DenseNet121()
# net = ResNeXt29_2x64d()
# net = MobileNet()
# net = MobileNetV2()
# net = DPN92()
# net = ShuffleNetG2()
# net = SENet18()
# net = ShuffleNetV2(1)
# net = EfficientNetB0()
# net = RegNetX_200MF()
# net = SimpleDLA()
```

```python
136     # Save checkpoint.
137     acc = 100.*correct/total
138     if acc > best_acc:
139         print('Saving..')
140         state = {
141             'net': net.state_dict(),
142             'acc': acc,
143             'epoch': epoch,
144         }
145         if not os.path.isdir('checkpoint'):
146             os.mkdir('checkpoint')
147         torch.save(state, './checkpoint/ckpt.pth')
148         best_acc = acc
```

  - ✓ Create a “checkpoint” folder and address it to line 147 for saving the results in “.pth” format after 200 epochs.

VGG Without M. P.

- Modifying the VGG Model
  - ✓ @ "vgg.py", By modifying this part of the code, we removed All Max Pooling Layers.




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

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

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

    def forward(self, x):
        out = self.features(x)
        out = out.view(out.size(0), -1)
        out = self.classifier(out)
        return out

    def _make_layers(self, cfg):
        layers = []
        in_channels = 3
        for x in cfg:
            layers += [nn.Conv2d(in_channels, x, kernel_size=3, padding=1),
                       nn.BatchNorm2d(x),
                       nn.ReLU(inplace=True)]
            in_channels = x
        layers += [nn.AdaptiveAvgPool2d((1, 1))]
        return nn.Sequential(*layers)

def test():
    net = VGG('VGG16')
    x = torch.randn(2, 3, 32, 32)
    y = net(x)
    print(y.size())
```

VGG Without M. P.

- Code Structure

Cloning Kuangliu pytorch_cifar Repository And applying previous slide’s modification

Command to Train VGG 16 Without M. P. for 200 epochs

```
!git clone https://github.com/kuangliu/pytorch-cifar.git
```

```
Cloning into 'pytorch-cifar'...
remote: Enumerating objects: 382, done.
remote: Counting objects: 100% (382/382), done.
remote: Compressing objects: 100% (182/182), done.
remote: Total 382 (delta 209), reused 355 (delta 197), pack-reused 0
Receiving objects: 100% (382/382), 77.42 KiB | 5.53 MiB/s, done.
Resolving deltas: 100% (209/209), done.
```

```
%cd /content/pytorch-cifar
```

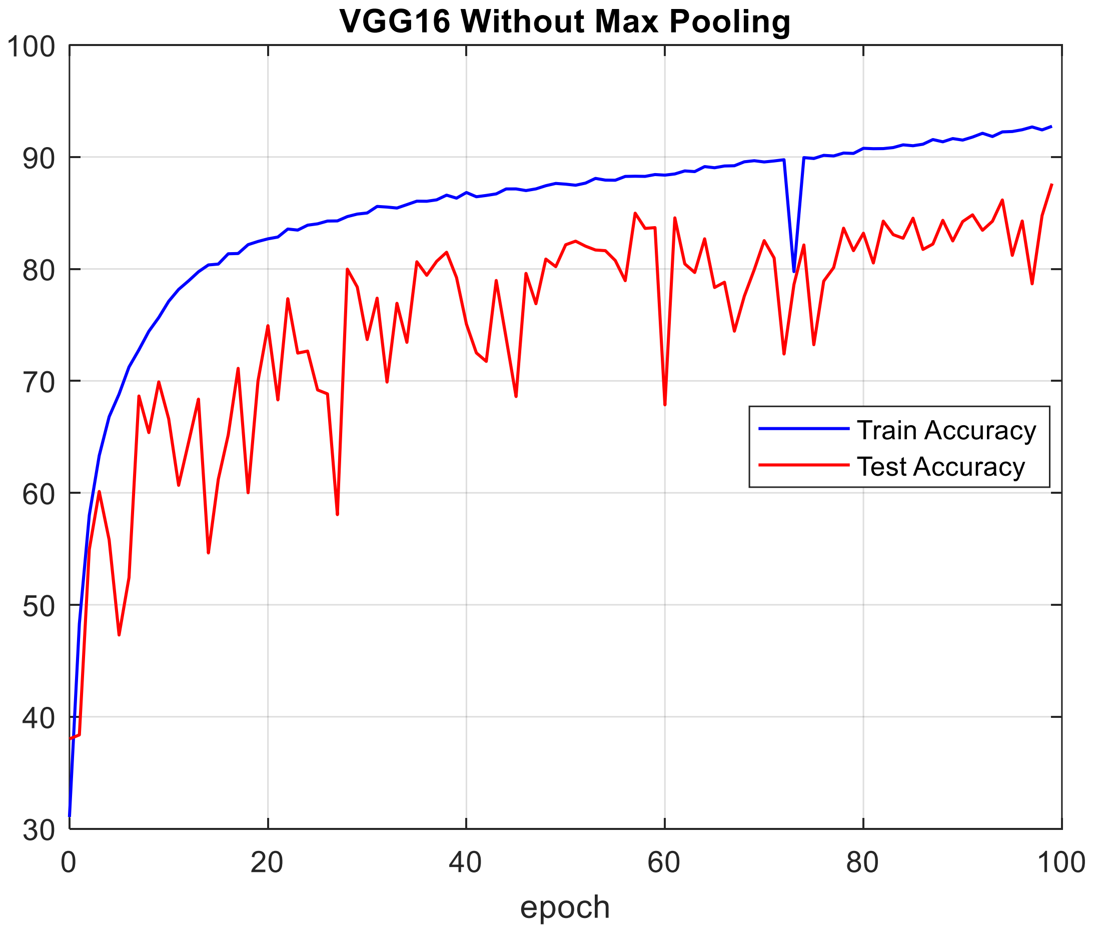
```
/content/pytorch-cifar
```

```
# Start training with:
!python main.py

# You can manually resume the training with:
!python main.py --resume --lr=0.01
```

```
==> Preparing data..
Downloading https://www.cs.toronto.edu/~kriz/cifar-10-python.tar.gz to ./data/cifar-10-python.tar.gz
100% 170498071/170498071 [00:03<00:00, 52056434.58it/s]
Extracting ./data/cifar-10-python.tar.gz to ./data
Files already downloaded and verified
==> Building model..

Epoch: 0
```

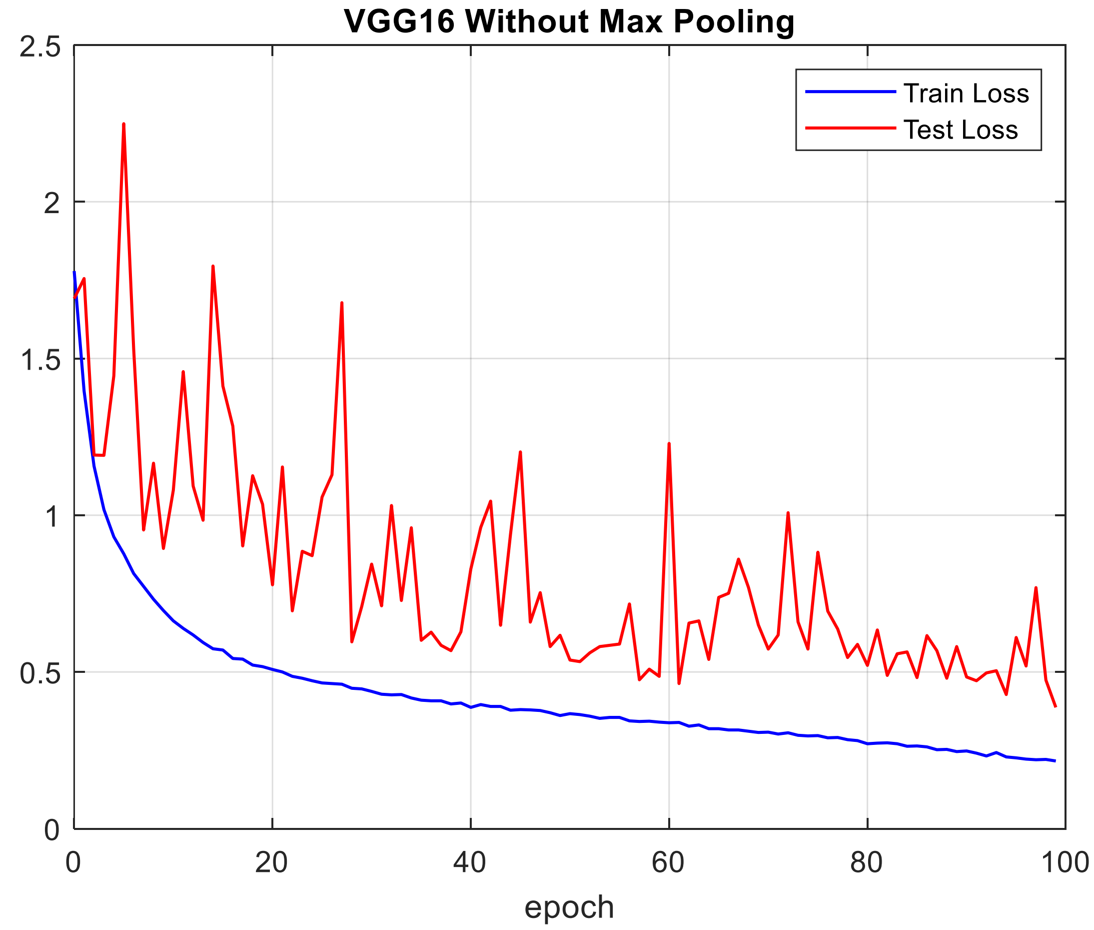
VGG Without M. P.

- Output
  - ✓ After the Iteration of 200 epochs, the results are obtained as follows:



VGG Without M. P.

- Output
  - ✓ After the Iteration of 200 epochs, the results are obtained as follows:



VGG Without M. P.

- Screenshots of Iterated Epochs

```
Epoch: 197
 [================================================================>]  Step: 43ms | Tot: 25s967ms | Loss: 0.003 | Acc: 100.000% (50000/50000) 391/391
 [================================================================>]  Step: 30ms | Tot: 2s978ms | Loss: 0.191 | Acc: 94.000% (9400/10000) 100/100

Epoch: 198
 [================================================================>]  Step: 44ms | Tot: 25s953ms | Loss: 0.003 | Acc: 100.000% (50000/50000) 391/391
 [================================================================>]  Step: 30ms | Tot: 2s967ms | Loss: 0.192 | Acc: 94.040% (9404/10000) 100/100

Epoch: 199
 [================================================================>]  Step: 43ms | Tot: 25s978ms | Loss: 0.003 | Acc: 100.000% (50000/50000) 391/391
 [================================================================>]  Step: 30ms | Tot: 2s964ms | Loss: 0.192 | Acc: 94.060% (9406/10000) 100/100
==> Preparing data..
Files already downloaded and verified
Files already downloaded and verified
==> Building model..
==> Resuming from checkpoint..
```

VGG Without M. P.

- Screenshots of Iterated Epochs

```
Epoch: 291
 [================================================================>]  Step: 44ms | Tot: 25s935ms | Loss: 0.032 | Acc: 99.384% (49692/50000) 391/391
 [================================================================>]  Step: 29ms | Tot: 2s964ms | Loss: 0.269 | Acc: 91.520% (9152/10000) 100/100

Epoch: 292
 [================================================================>]  Step: 44ms | Tot: 25s950ms | Loss: 0.020 | Acc: 99.720% (49860/50000) 391/391
 [================================================================>]  Step: 29ms | Tot: 2s975ms | Loss: 0.254 | Acc: 92.260% (9226/10000) 100/100

Epoch: 293
 [================================================================>]  Step: 43ms | Tot: 25s921ms | Loss: 0.017 | Acc: 99.692% (49846/50000) 391/391
 [================================================================>]  Step: 29ms | Tot: 2s962ms | Loss: 0.252 | Acc: 92.310% (9231/10000) 100/100

Epoch: 294
 [================================================================>]  Step: 44ms | Tot: 25s930ms | Loss: 0.010 | Acc: 99.900% (49950/50000) 391/391
 [================================================================>]  Step: 29ms | Tot: 2s966ms | Loss: 0.229 | Acc: 92.910% (9291/10000) 100/100

Epoch: 295
 [================================================================>]  Step: 44ms | Tot: 25s949ms | Loss: 0.006 | Acc: 99.950% (49975/50000) 391/391
 [================================================================>]  Step: 29ms | Tot: 2s982ms | Loss: 0.223 | Acc: 93.100% (9310/10000) 100/100
```

ML – Final Project

Chapter5: Conclusion

Comparing the results and answering the ambiguities.

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