Final Project Report

사이버보안전공 1971063 김윤서

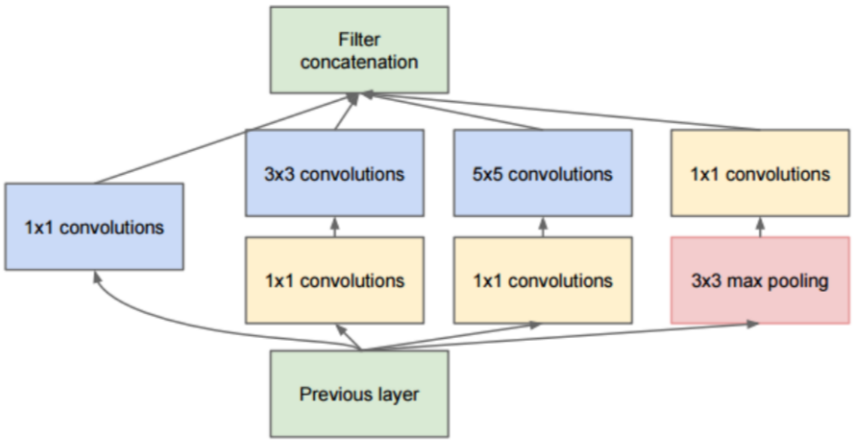
목차

---

1. Inception module 구현

1×1, 3×3 등의 작은 Convolutional layer 여러 개를 한 층에서 구성하는 형태를 취하는 Inception module 을 구현했다.

→ 여러 층의 Inception module을 구성함으로써, 전체적인 연산량을 줄이고 정확도를 높여주었다.



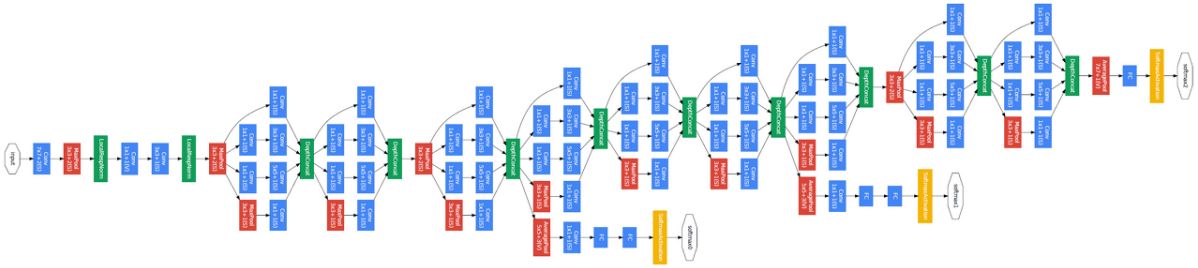
구현한 Inception module 그림 ↑

코드

```
class Inception(nn.Module):
    def __init__(self, in_planes, n1x1, n3x3red, n3x3, n5x5red, n5x5, pool_planes):
        super(Inception, self).__init__()
        # 1x1 conv branch
        self.b1 = nn.Sequential(
            nn.Conv2d(in_planes, n1x1, kernel_size=1),
            nn.BatchNorm2d(n1x1),
            nn.ReLU(True),
        )

        # 1x1 conv -> 3x3 conv branch
        self.b2 = nn.Sequential(
            nn.Conv2d(in_planes, n3x3red, kernel_size=1),
            nn.BatchNorm2d(n3x3red),
            nn.ReLU(True),
            nn.Conv2d(n3x3red, n3x3, kernel_size=3, padding=1),
            nn.BatchNorm2d(n3x3),
            nn.ReLU(True),
        )

        # 1x1 conv -> 5x5 conv branch
        self.b3 = nn.Sequential(
            nn.Conv2d(in_planes, n5x5red, kernel_size=1),
            nn.BatchNorm2d(n5x5red),
            nn.ReLU(True),
            nn.Conv2d(n5x5red, n5x5, kernel_size=5, padding=2),
            nn.BatchNorm2d(n5x5),
            nn.ReLU(True),
            nn.Conv2d(n5x5, n5x5, kernel_size=5, padding=2),
            nn.BatchNorm2d(n5x5),
            nn.ReLU(True),
        )

        # 3x3 pool -> 1x1 conv branch
        self.b4 = nn.Sequential(
            nn.MaxPool2d(3, stride=1, padding=1),
            nn.Conv2d(in_planes, pool_planes, kernel_size=1),
            nn.BatchNorm2d(pool_planes),
            nn.ReLU(True),
        )

    def forward(self, x):
        y1 = self.b1(x)
        y2 = self.b2(x)
        y3 = self.b3(x)
        y4 = self.b4(x)
        return torch.cat([y1,y2,y3,y4], 1)
```

2. Inception module을 이용해 GoogLeNet 구현

이미지 인식에 좋은 성능을 내는 **GoogLeNet**을 참고하였다.



- 특징

→ 22개 층으로 구성

→ 위에서 정의한 Inception module을 여러 개 쌓아 모델 구현

- 모델 structure

type	patch size/ stride	output size	depth	#1×1	#3×3 reduce	#3×3	#5×5 reduce	#5×5	pool proj	params	ops
convolution	7×7/2	112×112×64	1							2.7K	34M
max pool	3×3/2	56×56×64	0								
convolution	3×3/1	56×56×192	2		64	192				112K	360M
max pool	3×3/2	28×28×192	0								
inception (3a)		28×28×256	2	64	96	128	16	32	32	159K	128M
inception (3b)		28×28×480	2	128	128	192	32	96	64	380K	304M
max pool	3×3/2	14×14×480	0								
inception (4a)		14×14×512	2	192	96	208	16	48	64	364K	73M
inception (4b)		14×14×512	2	160	112	224	24	64	64	437K	88M
inception (4c)		14×14×512	2	128	128	256	24	64	64	463K	100M
inception (4d)		14×14×528	2	112	144	288	32	64	64	580K	119M
inception (4e)		14×14×832	2	256	160	320	32	128	128	840K	170M
max pool	3×3/2	7×7×832	0								
inception (5a)		7×7×832	2	256	160	320	32	128	128	1072K	54M
inception (5b)		7×7×1024	2	384	192	384	48	128	128	1388K	71M
avg pool	7×7/1	1×1×1024	0								
dropout (40%)		1×1×1024	0								
linear		1×1×1000	1							1000K	1M
softmax		1×1×1000	0								

GoogLeNet 코드

```
class GoogLeNet(nn.Module):
    def __init__(self):
        super(GoogLeNet, self).__init__()
```

```
        self.pre_layers = nn.Sequential(
            nn.Conv2d(3, 192, kernel_size=3, padding=1),
            nn.BatchNorm2d(192),
            nn.ReLU(True),
        )

        self.a3 = Inception(192,  64,  96, 128, 16, 32, 32)
        self.b3 = Inception(256, 128, 128, 192, 32, 96, 64)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a4 = Inception(480, 192,  96, 208, 16,  48,  64)
        self.b4 = Inception(512, 160, 112, 224, 24,  64,  64)
        self.c4 = Inception(512, 128, 128, 256, 24,  64,  64)
        self.d4 = Inception(512, 112, 144, 288, 32,  64,  64)
        self.e4 = Inception(528, 256, 160, 320, 32, 128, 128)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a5 = Inception(832, 256, 160, 320, 32, 128, 128)
        self.b5 = Inception(832, 384, 192, 384, 48, 128, 128)

        self.avgpool = nn.AvgPool2d(8, stride=1)
        self.linear = nn.Linear(1024, 10)


    def forward(self, x):

        out = self.pre_layers(x)
        out = self.a3(out)
        out = self.b3(out)
        out = self.maxpool(out)
        out = self.a4(out)
        out = self.b4(out)
        out = self.c4(out)
        out = self.d4(out)
        out = self.e4(out)
        out = self.maxpool(out)
        out = self.a5(out)
        out = self.b5(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.linear(out)

        return out
```

GoogLeNet 결과 ⇒ 74%

```
[1,  2000] loss: 1.896
[1,  4000] loss: 1.560
[1,  6000] loss: 1.388
[1,  8000] loss: 1.223
[1, 10000] loss: 1.138
[1, 12000] loss: 1.066
[2,  2000] loss: 0.921
[2,  4000] loss: 0.892
[2,  6000] loss: 0.866
[2,  8000] loss: 0.797
[2, 10000] loss: 0.794
[2, 12000] loss: 0.737
Finished Training
Saved Trained Model
```

```
# Test
googlenet.load_state_dict(torch.load(PATH))
print_accuracy(googlenet, testloader)
```

```
Accuracy of the network on the 10000 test images: 74 %
```

3. MyNet 모델 정의

GoogLeNet 아키텍처를 기반으로 MyNet 모델을 구현하였다.

먼저 GoogLeNet 모델은 336 * 336 px 크기의 이미지 분류에 특화되어 있기 때문에,

1) 해당 데이터셋에 알맞게 **Inception 순서를 조정**하고,
2) **Conv layer 크기**를 아래와 같이 변경해주었다.

▼ MyNet 모델 구조

1. Conv2d
2. BatchNorm2d
3. ReLU
4. Inception
    - n1×1(48), n3×3(8), n5×5(16), pool(16)
5. Inception
    - n1×1(96), n3×3(16), n5×5(32), pool(32)
6. MaxPool2d
7. Inception

- n1×1(160), n3×3(256), n5×5(64), pool(64)

8. Inception
    - n1×1(256), n3×3(256), n5×5(128), pool(128)
9. Inception
    - n1×1(256), n3×3(256), n5×5(128), pool(128)
10. MaxPool2d
11. Inception
    - n1×1(256), n3×3(512), n5×5(128), pool(128)
12. Inception
    - n1×1(384), n3×3(384), n5×5(128), pool(128)
13. AvgPool2d
14. Linear

MyNet 코드

```
class MyNet(nn.Module):
    def __init__(self):
        super(MyNet, self).__init__()

        self.pre_layers = nn.Sequential(
            nn.Conv2d(3, 128, kernel_size=3, padding=1),
            nn.BatchNorm2d(128),
            nn.ReLU(True),        #활성화 함수 ReLU 사용
        )

        self.a3 = Inception(128,  32,  48,  64,  8, 16, 16)
        self.b3 = Inception(128,  64,  96, 128, 16, 32, 32)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a4 = Inception(256, 160,  96, 256, 16,  64,  64)
        self.b4 = Inception(544, 256, 128, 256, 64, 128, 128)
        self.c4 = Inception(768, 256, 128, 256, 64, 128, 128)

        self.maxpool = nn.MaxPool2d(3, stride=2, padding=1)

        self.a5 = Inception(768, 256, 256, 512, 64, 128, 128)
        self.b5 = Inception(1024, 384, 192, 384, 48, 128, 128)
```

```
        self.avgpool = nn.AvgPool2d(8, stride=1)  # Average Pooling 사용
        self.linear = nn.Linear(1024, 10)


    def forward(self, x):

        out = self.pre_layers(x)
        out = self.a3(out)
        out = self.b3(out)
        out = self.maxpool(out)
        out = self.a4(out)
        out = self.b4(out)
        out = self.c4(out)
        out = self.maxpool(out)
        out = self.a5(out)
        out = self.b5(out)
        out = self.avgpool(out)
        out = out.view(out.size(0), -1)
        out = self.linear(out)

        return out
```

MyNet 결과 ⇒ 76%

```
[1,  2000] loss: 1.853
[1,  4000] loss: 1.514
[1,  6000] loss: 1.327
[1,  8000] loss: 1.168
[1, 10000] loss: 1.084
[1, 12000] loss: 0.970
[2,  2000] loss: 0.898
[2,  4000] loss: 0.849
[2,  6000] loss: 0.816
[2,  8000] loss: 0.785
[2, 10000] loss: 0.752
[2, 12000] loss: 0.720
Finished Training
Saved Trained Model
```

```
[36]: # Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 76 %
```

4. 성능 개선 일지

▾ * hyper-parameter 목록

1. **learning rate** (lr)=0.001, 0.0005, 0.0001, ...
2. **Batch size**=4,8,16,32
3. **Epochs**=2,4, ...
4. Network architectures: googLeNet
5. Activation functions: RELU 사용
6. Loss function: Sotfmax cross entropy

시도 1

*epochs=**4*** 로 변경

- 결과 ⇒ **82%**

```
epochs = 4

# Train
train(myNet, trainloader, epochs, criterion, optimizer, PATH)
```

```
[1,  2000] loss: 1.866
[1,  4000] loss: 1.531
[1,  6000] loss: 1.323
[1,  8000] loss: 1.180
[1, 10000] loss: 1.071
[1, 12000] loss: 1.008
[2,  2000] loss: 0.885
[2,  4000] loss: 0.842
[2,  6000] loss: 0.838
[2,  8000] loss: 0.747
[2, 10000] loss: 0.751
[2, 12000] loss: 0.721
[3,  2000] loss: 0.639
[3,  4000] loss: 0.615
[3,  6000] loss: 0.597
[3,  8000] loss: 0.595
[3, 10000] loss: 0.597
[3, 12000] loss: 0.586
[4,  2000] loss: 0.487
[4,  4000] loss: 0.469
[4,  6000] loss: 0.493
[4,  8000] loss: 0.471
[4, 10000] loss: 0.485
[4, 12000] loss: 0.482
Finished Training
Saved Trained Model
```

```
# Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 82 %
```

시도 2

*lr=**0.0001*** 로 변경

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(myNet.parameters(), lr=0.0001, momentum=0.9)

PATH = './my_net.pth'
epochs = 4
```

- 결과: **75%** 로 더 낮아짐
  → learning rate는 0.001로 유지하도록 한다.

```
# Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 75 %
```

시도 3

*batch size=**16*** 으로 변경

```
trainset = torchvision.datasets.CIFAR10(root='./data', train=True,
                                        download=True, transform=transform)
trainloader = torch.utils.data.DataLoader(trainset, batch_size= 16,
                                          shuffle=True, num_workers=2)

testset = torchvision.datasets.CIFAR10(root='./data', train=False,
                                       download=True, transform=transform)
testloader = torch.utils.data.DataLoader(testset, batch_size= 16,
                                         shuffle=False, num_workers=2)
```

- 결과 ⇒ **83%**

  → 변화 적음

  → *batch size*를 더 바꿔보며 적절한 값을 찾자.

```
# Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 83 %
```

시도 4

batch size=**8**

- 결과 ⇒ **83%**

```
# Train
train(myNet, trainloader, epochs, criterion, optimizer, PATH)
```

```
[1,  2000] loss: 1.542
[1,  4000] loss: 1.112
[1,  6000] loss: 0.911
[2,  2000] loss: 0.742
[2,  4000] loss: 0.700
[2,  6000] loss: 0.649
[3,  2000] loss: 0.530
[3,  4000] loss: 0.525
[3,  6000] loss: 0.491
[4,  2000] loss: 0.400
[4,  4000] loss: 0.407
[4,  6000] loss: 0.396
Finished Training
Saved Trained Model
```

```
# Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 83 %
```

⇒ *batch size*를 4, 8, 16 으로 설정했을 때 결과 거의 비슷

→ *batch size*를 좀 더 늘려보자.
→ *epoch*=2 →4로 바꿨을 때도 성능이 올라갔으므로 epoch도 좀 더 바꿔보자.

시도 5

batch size=**32**

- 결과 ⇒ **81%**

```
# Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 81 %
```

⇒ *batch size* 높여도 성능 변화 크지 않으므로, *batch size= 8*로 유지하도록 함

시도 6

*epochs = **6***
*batch_size **= 8***
*lr = **0.0005*** 조합

```
criterion = nn.CrossEntropyLoss()
optimizer = optim.SGD(myNet.parameters(), lr=0.0005, momentum=0.9)

PATH = './my_net.pth'
epochs = 6
```

- 결과 ⇒ **84%**

```
# Test
myNet.load_state_dict(torch.load(PATH))
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 84 %
```

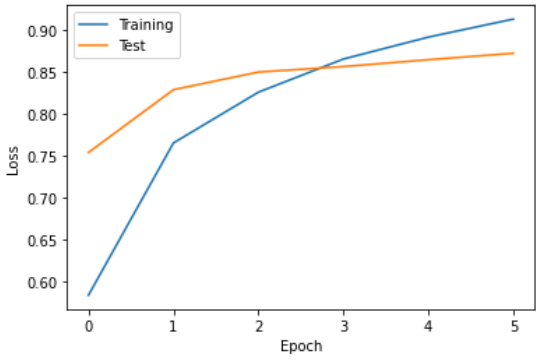
시도 7

*epochs = **6***
*batch size = **8***
*lr = **0.001*** 조합

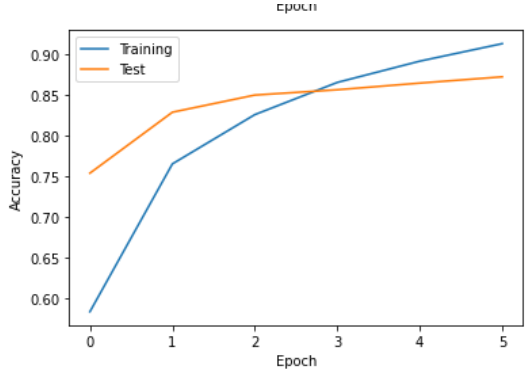
- 결과 ⇒ **87%**

```
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 87 %
```



Loss 그래프



Accuracy 그래프

5. 최종 결과

⇒ 정확도 **87%** 도출

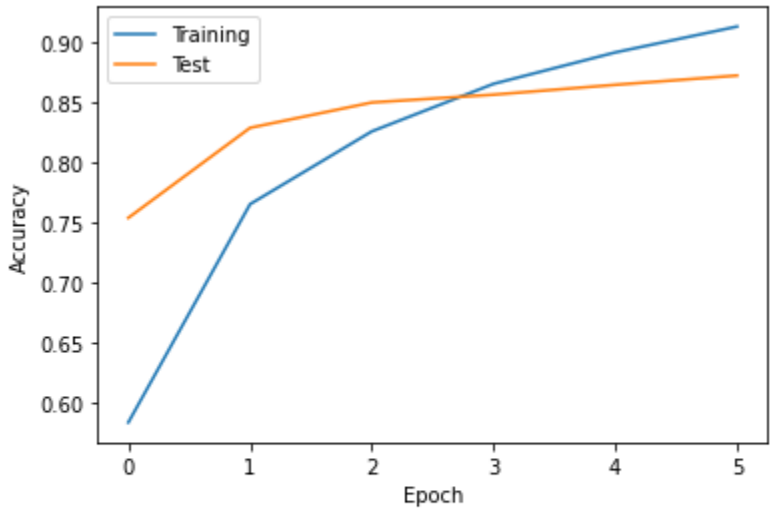
```
[23]
    myNet, loss_hist, metric_hist = train_val(myNet, params_train)
```

```
train loss: 0.146752, val loss: 0.089665, accuracy: 75.39
----------
train loss: 0.086928, val loss: 0.063040, accuracy: 82.88
----------
train loss: 0.065011, val loss: 0.055650, accuracy: 84.99
----------
train loss: 0.050329, val loss: 0.052089, accuracy: 85.64
----------
train loss: 0.039856, val loss: 0.050668, accuracy: 86.46
----------
train loss: 0.031717, val loss: 0.049372, accuracy: 87.23
----------
```

사용한 hyper-parameter 요약:

- Loss 함수: CrossEntropy
- Optimizer: Gradient descent with lr = 0.001 (learning rate)
- Epochs= 6
- Batch_size = 8
- Activation 함수: ReLU

최종 결과 시각화 (learning curve)




```
print_accuracy(myNet, testloader)
```

```
Accuracy of the network on the 10000 test images: 87 %
```